INESC TEC

LASCA

Large Scale Computing with Autoencoders - Application to Power Systems

PTDC/EEA-EEL/104278/2008

LASCA is a research project on a new procedure to speed up and improve convergence of meta-heuristics in Large Scale problem optimization.

Final Report
Oct. 2013

INESCTEC
TECHNOLOGY & SCIENCE
ASSOCIATE LABORATORY

COORDINATED BY
INESCPORTO
PORTUGAL

INESCTEC

Final report describing activities, experiments, tests and results from the work developed at INESC Technology and Science, Associate Laboratory in the framework of the LASCA project funded by FCT – the Foundation for Science and Technology.

Contents

Table of Figures

LASCA - Large Scale Computing with Autoencoders - Application to Power Systems

PTDC/EEA-EEL/104278/2008

Project LASCA / Final Report

1 EXECUTIVE SUMMARY

The project LASCA aimed at testing a novel strategy to address large scale optimization problems, when using metaheuristics as a solver. It is known that the meta-heuristics behaviour (such as in evolutionary algorithms) degrades quite considerably with the dimensional growth of the search space. The conjecture behind the LASCA approach, therefore, was that a reduction in the search space dimension, even at the cost of some loss in precision, would lead, at least in some classes of problems, to a speed up in convergence of some meta-heuristic and, in a favourable scenario, to a convergence with higher precision (for instance, from escaping to getting trapped in local optima).

The strategy put to test was the following:

1. **Given a problem, run a meta-heuristic optimization algorithm for some generations, and collect data about the progress of the search.**
2. **Use these data to train an autoencoder**
3. **Use the autoencoder to obtain a projection of the search into a smaller dimension space**
4. **Organize the progress of the search for the optimum in the new space**
5. **At some point, return to the higher dimension space and tune up the solution.**

The project success involved a certain level of risk: the strategy might prove not valid, or not workable, or not good enough and not compensating for the effort put in its implementation. The development of the project proved this assessment of the risk involved to be correct. But, finally, success was achieved and the project could be completed with extremely positive results.

As an extremely valuable complementary result (nevertheless crucial to the success of the project), one could derive new principles and a new procedure to train autoassociateive neural networks in a much more efficient way than reported in the state-of-the-art literature – based on principles related to Information Theoretic Learning and resorting to unsupervised training of the first half of the neural network instead of the classical supervised training of the whole network. This achievement was not predicted in the project proposal.

In summary, the project demonstrated that the new solving strategy devised may produce very good results. Therefore, it has the potential to become one important weapon in the toolkit of the large scale programming solver. Furthermore, a new and more efficient process of training autoencoders was created.

The search for and validation of these scientific and very valuable results allowed the production of contributions for internations journal papers that exceed what was predicted in the project proposal (5 against 4). Papers in conferences were naturally produced. The work contributed strongly to the contents of one PhD thesis and moderately to a second PhD thesis.

2 INTRODUCTION

The project LASCA aimed at solving large scale problems in power systems with the application of autoassociative neural networks (autoencoders). This Final Report provides the overview on the research activities developed during the fulfilment of LASCA project. Therefore, this report provides the holistic view on the activities performed, and the correspondent alignment with the syllabus firstly proposed within the proposal.

Autoencoders are effective in providing a useful tool to compress and expand information in various domains. One of the original ideas explored in this project relates with the employment of an evolutionary optimization metaheuristic into a reduced space S' (obtained with autoencoders' ability to reduce dimensionality), whose evolution is controlled in the original space S. This way, the autoencoders map the transition between the two spaces S and S'. The research conducted led to the proposal of a new technique, named "LASCA". This technique was applied to benchmark optimization functions, and to the Hydro-Wind coordination problems. The results obtained revealed a potential for the achievement of better solutions than simply applying a meta-heuristic (the Evolutionary Particle Swarm Optimization EPSO was chosen for testing) in the original search space.

The progress of the project soon demonstrated that the quality of the training of the autoencoders has a strong effect in the quality of the results achieved. Therefore, all efforts were primarily put in this intermediary research topic, which assumed a degree of importance not foreseen in the beginning – although there was a provision for this in the original project proposal.

This strategy for conducting the project was discussed and validated with the external expert invited to accompaign the project: Prof. Mohamed El-Sharkawi, from the University of Washington (USA).

This important research topic pursued in project LASCA concerned the development of a new training procedure for autoassociative neural networks using Information Theoretic Learning (ITL) concepts, its validation in test problems and its application to power system problems. Four main types of ITL autoencoders where developed: i) maximizing the Renyi's Quadratic Entropy at bottleneck layer, ii) maximizing the Quadratic Mutual Information (QMI) between the first and bottleneck layers with Cauchy Schwartz Estimation, iii) maximizing QMI between the first and bottleneck layers with Euclidean Distance Estimation, iv) minimizing QMI between all combinations of different neurons at bottleneck layer. Using adequate datasets for train, test and validation, alongside with the employment of adequate statistical tests, it was found that the quality of solutions provided by ITL methods was better than the classic Backpropagation (PROP) method, although ITL methods demand higher processing time. Among the ITL

methods, a similar level of solution quality was obtained, and the minimization of QMI between input and bottleneck layers with Cauchy Schwartz Estimation revealed the best processing time, justifying its selection for further applications in power system problems.

The working papers produced are provided in the next chapters of this report. The scientific contributions attained within this project are contained in the literature published, namely papers and theses submitted to the University of Porto and the Federal University of Maranhão, Brazil. The most relevant literature is provided in the annexes of this document. The project main objectives and results are detailed in the following sections, demonstrating that the Research Plan originally submitted was adequately followed.

3 PROJECT PLANNING AND ITS COMPLETION

This section summarizes the main tasks initially planned for the development of project LASCA as well as how they were developed. All tasks were duly completed.

3.1 State of the art in autoencoders

This task was devoted to a thorough examination of research published in the domain of autoencoders in order to extract useful information and lessons for the project. It was also devoted to the study of both autoencoders and the evolutionary algorithms, especially EPSO, by the new researchers.

The need for this task was identified by realizing that autoassociative neural networks is still a somewhat exotic field and that the new researchers recruited to the project team would likely not be familiar with either this topic or with the Evolutionary Particle Swarm Optimization algorithm (EPSO). Obviously, no scientific result was expected from this task.

3.2 Training autoencoders

This task was to be devoted to developing and implementing efficient processes to train and tune autoencoders. It demands experimenting a set of alternatives from backpropagation to evolutionary algorithms, from batch training to incremental layer training, from adopting the classical MSE (minimum square error) to MEE (minimum error entropy) criteria.

In fact, this task became, in terms of effort, the main task of the project, due to the difficulties it became necessary to overcome. Its completion produced an extremely valuable result in terms of a new procedure to accurately train autoencoders, based on splitting the neural network along its middle layer (of reduced dimension) and applying an unsupervised training procedure the first half (from input to middle layer). This

unsupervised traing must be conducted under some criterion based on Information Theoretic Learning concepts, such as maximizing the mutual information between the input and output vectors.

3.3 Software platform development

This task was devoted to building a software platform to integrate EPSO with autoencoders for optimization. This included a suitable and efficient autoencoder training procedure inside the optimization process. The software platform was customized to enable the running of Power System problems and its performance optimized taking this type of application in account.

This task progressed naturally in parallel with the scientific tasks, as the test and validation cases were built up.

3.4 Hybrids autoencoder – EPSO

This task was devoted to studying the best way to embed autoencoders in the EPSO evolutionary algorithm to maximize the optimization efficiency. A practical solution was achieved but it must be recognized that this objective showed dependence on the type of problems being addressed. No scientific results were expected from this task, only an improvement in the software platform.

3.5 Tests in real world large scale Power System

This task included 1.The selection of specific problems in the Power Systems area presenting aspects that could benefit from a hybrid evolutionary/autoencoder solving strategy. 2. The adaptation of the softwares and the development of specific software to solve the Power System problems selected. 3. The analysis of the results.

The task was actually enlarged by adding to it a preliminary phase: the selection of a set of standard mathematical test functions to verify the performance of the LASCA strategy, before passing to power system problems.

The power system problems selected to test the efficiency and validity of the new techniques were the following:

1. Estimation of breaker status under wrong or missing information in meshed networks
2. Wind-hydro coordination in systems with cascading reservoirs

The first problem was especially selected to verify, in a difficult real world problem, the validity of the new autoassociative neural network training procedures.

The second problem was designed in two versions (smaller and larger) to test the efficiency of the LASCA strategy.

The choice of these problems was made after discussion with the project advisor (Prof. El-Sharkawi).

3.6 Dissemination

This task was included in the project mainly to take in account the effort of

1. Writing conference and journal papers and submiting them for publication.
2. Preparing a website with a public information about the project and its advancement.
3. Participating in scientific events to present the work.

The objectives foreseen were duly completed. The website related to the project is referenced by

http://lasca.inescporto.pt.

3.7 Final Report

This task was devoted to the writing of the final project report and is accomplished.

4 ADVICE FROM THE EXTERNAL PROJECT CONSULTANT

As foreseen in the project proposal, Prof. Mohamed El-Sharkawi, from the USA, was invited as Project Consultant.

Prof. El-Sharkawi is a renowned personality and gave a valuable contribution to the project. He is an IEEE Fellow and Director of the CIA-LAB (Computational Intelligence Applications Laboratory) in the Department of Electrical Engineering of the University of Washington. He is a member of the administrative committee of the IEEE Neural Networks Council representing the Power Engineering Society and Video Tutorial Chair of the IEEE Continuing Education Committee.

Prof. El-Sharkawi visited INESC TEC twice during the duration of the project (in 2012 and in 2013), each time for a week, besides cooperating regularly in the discussion of the project progress via the normal electronic communication means. He produced two reports that are added to this Final Report as Annexes.

In summary, he not only gave valuable advice but agreed with the project development strategy put in place.

5 RELATION WITH OTHER PROJECTS SUPPORTED BY FCT

This project LASCA has strong connections with two projects funded by FCT:

POSC/EEA-ESE/60980/2004
EPSO - Evolutionary Particle Swarm Optimization, a new meta-heuristic with applications in power systems.

This project ended in 2008 with a strong success: it allowed the validation of a new meta-heuristic denoted EPSO, which proved more efficient in benchmarking tests that the classical PSO (Particle Swarm Optimization).

This EPSO algorithm, running in the platform developed in this project, was an essential tool in the LASCA strategy.

PTDC/EEA-EEL/105261/2008
GEMS - Gross errors and missing signals: new concepts in power system state estimation

This project ended in 2012 also with success. It was mainly devoted to new models in power system state estimation. It benefited, at some point, from the results achieved in the LASCA project from the new processes to train autoencoders and provided one of the two real world applications selected to test and validate the components of the LASCA strategy.

6 CONCLUSIONS

The promised results of the project were:

1. A methodology to include autoencoders in evolutionary optimization processes
2. A software platform
3. New and more efficient models to solve selectd Power System problems
4. Papers submitted and published in international conferences and journals
5. Contributions to PhD and MSc. theses

This report confirms that all objectives were met and the results even exceeded expectations in some points.

In summary:

- The project met all goals with success.
- Valuable scientific results were generated.
- Publications were produced as predicted, namely in the most prestigious international journals in the Electric and Electronic Engineering field: the IEEE Transactions.
- The results were integrated in two PhD Theses.

The effort to publish results from the project continued even after its ending date, because there is still material of very interesting quality to be submitted to the scientific community. Some papers are undergoing a process of submission to journals and yet some more are foreseen, at least to important conferences.

The PI is grateful to the FCT for the funding provided, which was essencial to guarantee the conditions to proceed with the research reported.

Porto, October 31, 2013

Vladimiro Miranda, IEEE Fellow
Full Professor, Faculty of Engineering of the University of Porto, Portugal
Researcher at INESC TEC – INESC Technology and Science, Portugal

Publications

The following publications were produced incorporating results from the project work.

Papers in journals

[1] Jakov Krstulovic, Vladimiro Miranda, Antônio Simões Costa, Jorge Correia Pereira, **Towards an auto-associative topology state estimator**, *IEEE Transactions on Power Systems*, vol.28, no.3, pp.3311-3318, Agosto, 2013.

[2] Vladimiro Miranda, Adriana Castro, **Diagnosing faults in power transformers with autoassociative neural networks and mean shift.**, *IEEE Transactions on Power Delivery*, vol.27, no.3, pp.1350-1357, Julho, 2012.

[3] Vladimiro Miranda, Jakov Krstulovic, Hrvoje Keko, Cristiano Moreira, Jorge Pereira, **Reconstructing missing data in State Estimation with autoencoders**, *IEEE Transactions on Power Systems*, vol.27, no.2, pp.604-611, Maio, 2012.

Beside these already published papers, other papers are being prepared/submitted for publication in an international journal:

[4] Vladimiro Miranda, Vera Palma Ferreira and Joana Hora Martins, “Optimizing large scale problems in a reduced space mapped by autoencoders”

[5] Cátia S. P. Silva, Jakov Krstulovic, Joana H. Martins, Vera Palma, Vladimiro Miranda and José C. Príncipe, “Extracting topology information from electric measurements: a model comparison”

Papers in international conferences

[1] Jean Sumaili, Vladimiro Miranda, Liviane Rego, Adamo Santana, Renato Francês, **A densification trick using mean shift to allow demand forecasting in special days with scarce data**, *Proceedings of ISAP 2013 - ISAP 2013 - International Conference on Intelligent Systems Applications in Power Systems*, Tokyo, Japan, Julho, 2013.

[2] Vladimiro Miranda, Jakov Krstulovic, Joana Hora, Vera Palma Ferreira, José Carlos Príncipe, **Breaker status uncovered by autoencoders under unsupervised maximum mutual information training**,

Proceedings of ISAP 2013 - ISAP 2013 - International Conference on Intelligent Systems Applications in Power Systems, Tokyo, Japan, Julho, 2013.

[3] Jakov Krstulovic, Vladimiro Miranda, Hrvoje Keko, Jorge Correia Pereira, **Descoberta da topologia do sistema na ausência de sinais com redes neurais autoassociativas**, *Proceedings of SBSE 2012 - IV Simpósio Brasileiro de Sistemas Elétricos*, Goiás, Brasil, Maio, 2012

[4] Vladimiro Miranda, Shigeaki Leite Lima, Adriana G. Castro, Osvaldo Saavedra, **Redes Neurais Autoassociativas Aplicadas ao Diagnóstico de Falhas em Transformadores de Potência**, *Proceedings of SEPOPE 2012 - XII SIMPÓSIO DE ESPECIALISTAS EM PLANEJAMENTO DA OPERAÇÃO E EXPANSÃO ELÉTRICA*, Rio de Janeiro, Brasil, Maio, 2012.

[5] Adriana Garcez Castro, Vladimiro Miranda, Shigeaki Leite Lima, **Transformer fault diagnosis based on autoassociative neural networks** , *ISAP 2011 - 16th International Conference on Intelligent System Applications to Power Systems*, Hersonissos, Grécia, Setembro, 2011.

PhD Theses

The following PhD theses incorporated results from the project work:

Shigeaki Leite de Lima

Incipient failure diagnosis and decision making in power transformers (in Portuguese – *Diagnóstico de falhas incipientes e tomada de decisão em transformadores de potência*)

Thesis submitted and approved within the Post-graduation Program in Electrical Engineering at the Federal University of Maranhão, São Luís (MA), Brazil, September 2013.

The PI was officially co-supervisor of the thesis.

Jakov Krstulovic Opara

A new paradign for power system state estimation (provisional title)

Thesis to be submitted, likely until the 31st of December, 2013, to FEUP (the Faculty of Enginnering of the University of Porto), within the Doctoral Program in Electrical and Computer Engineering.

The PI is officially the supervisor.

Both thesis incorporated, in different manner, the new training procedure developed for autoassociative neural networks.

External Consultant Reports

UNIVERSITY OF WASHINGTON
DEPARTMENT OF ELECTRICAL ENGINEERING, Box 352500
SEATTLE, WASHINGTON 98195-2500

Mohamed A. El-Sharkawi, Professor
Smart Energy Laboratory (SEL)
Computational Intelligence Applications (CIA) Laboratory

Telephone: *(206) 685-2286*
Electronic Mail: *elsharkawi@ee.washington.edu*
Home Page: *http://www.SmartEnergyLab.com*
Home Page: *http://www.cialab.org*

September 14, 2012

Progress review report on INESC project on "Large Scale Computing with Autoencoders - Application to Power System"

One of the most disabling problems in ultra-large-scale (ULS) systems is the enormous number of variables that needs to be processed. In the area of adaptive optimization, such large number of variables cannot be used and sub-optimal solutions are often reached by gross approximation in the system size or the number of variables to be considered. In a number of cases, the sub-optimal solution is far from the optimal and takes long time to compute. Therefore it cannot be adaptive for on-line applications.

The scope of this project is to explore the potential of using auto-encoder in feature extractions that allows the use of small set of features to perform adaptive optimization, and then scale the solution to the original ULS system. This high risk basic research can have a tremendous impact if the conjecture of the investigator is confirmed.

The research team at INESC has already developed several encoders using different techniques. One of them is quite novel as the encoder is split into two networks that are trained differently; one is unsupervised and the other is supervised. The results are promising and this work by itself is an achievement. The researchers are currently exploring different topologies and are testing the encoder for memorization and saturation. In addition, information on the performance index will be included to identify any enhancement in the encoder performance.

The principal investigator, Dr. Vladimiro Miranda, is one of the leading world experts in neural networks and data compression. He has three energetic researchers forming the team on this project. Although this research is high risk with no guarantee for success, I expect them to deliver several novel technologies that can be beneficial for future work.

The research strategy of the team is very good and I agree with it. It is beneficial to concentrate on the encoder first before the application to hydro-wind coordination is tackled.

Mohamed A. El-Sharkawi
Professor

UNIVERSITY OF WASHINGTON
DEPARTMENT OF ELECTRICAL ENGINEERING, Box 352500
SEATTLE, WASHINGTON 98195-2500

Mohamed A. El-Sharkawi, Professor
Smart Energy Laboratory (SEL)
Computational Intelligence Applications (CIA) Laboratory

Telephone: *(206) 685-2286*
Electronic Mail: *elsharkawi@ee.washington.edu*
Home Page: *http://www.SmartEnergyLab.com*
Home Page: *http://www.cialab.org*

June 12, 2013

Progress review report on INESC project on "Large Scale Computing with Autoencoders - Application to Power System"

This is my second visit to evaluate INESC progress on this project. In my first visit, I found that the research team had developed several encoders with some successful results. The recommendation at that time was to explore different topologies, enhance encoder performance, and test the encoder in wind-hydro coordination problem.

In this second visit, I am delighted that the research team was quite successful in designing effective encoders. Several designs, structures and training techniques were employed with promising results. In addition, the research team explored the use of the encoders in several important applications, two of them are listed below.

Hydro-wind coordination

This is one of the challenging problems associated with the stochastic production of wind energy systems. Because wind is varying, there are times when the grid is having surplus or deficit in energy. In these situations, hydro system can be used to store energy through pump storage or increase their energy production. This is a nonlinear optimization problem with several constraints. Solving this problem using the original data space is very hard as the scale and complexity of the problem are prohibitive. However, the team showed that the encoder developed for this problem can be used to provide an optimum solution to the hydro-wind coordination problem. The performance function they used is based on cost of energy from various resources, constraints imposed on the operation of the system, several penalty factors, and wind forecasting.

The successful results in this problem is highly encouraging and, in the future, the team can even extend the optimization procedure to include environmental factors such as irrigation constraints and fish sensitivity to nitrogen. When successful, the team at INESC would have solved on of the major problems facing utilities with hydro based systems and large scale wind turbines.

Estimation of Breaker Status

Another nice application for the autoencoder is to estimate the status of the circuit breakers in large scale system. INESC team has developed an auto association encoder for both closed and open status of breakers based on regional monitoring. The results shows that the accuracy is very high even when the currents are not monitored; the maximum failure rate is less than 2%.

For future addition, the research team may want to compare their technique to other methods and recommend the suitable solution based on the size and complexity of the power system. In addition, they may want to consider applying their technique to the distribution network, which is in great need for effective situation awareness methods.

In conclusion, based on what I have seen, I believe that the research team under the leadership of Dr. Vladimiro Miranda has achieved very promising results. Moreover, they have done more work than what is expected in this project. I would highly encourage the team to publish their research work and keep extending their work after this project is finished.

Sincerely,

Mohamed A. El-Sharkawi
Professor

Application of autoassociative neural networks to solve large scale problems in power systems

PTDC/EEA-EEL/104278/2008

Report LASCA / R1

7 Abstract

This paper synthetizes the work conducted under the scope of project LASCA concerning the application of autoassociative neural networks (autoencoders) to solve large scale problems in power systems. Autoencoders have been shown effective in providing a useful tool to compress and expand information in various domains. This work explores the autoencoders' ability to reduce dimensionality by applying an evolutionary optimization metaheuristic into a space of reduced dimension. This idea was firstly pursued with the goal of reducing computational effort, which is known to be significantly high for large scale problems. The methodology "Hybrid", firstly proposed in (Costa, 2008), was implemented. This methodology allows an evolutionary metaheuristic to evolve in a reduced dimension space S', controlling its evolution in the original space S. The results obtained showed that this approach does not allow the achievement of a computational effort reduction and that it has the potential to achieve better solutions in some case studies.

8 1 Introduction

The optimization of large scale problems with evolutionary algorithms is an adequate process to represent realistic features of real world problems. However, solving problems in high dimensional spaces is both demanding in computing resources and difficult on convergence into satisfactory solutions. These drawbacks usually lead to early termination of runs, inducing the metaheuristics' performance to be below than desirable.

The problem of dimensionality reduction has been addressed with applications in clustering and in image processing. One important technique is Principal Component Analysis (PCA) (Jolliffe, 2002), which is a mathematical procedure that projects the data into a linear subspace: data is multiplied with the

eigenvectors from the sample covariance matrix, from where each point is represented by its coordinates along the directions of greatest variance in the data set.

One research topic that has not been widely explored in literature, and that is explored in this work, is the combination of dimensionality reduction techniques as a general optimization tool for large scale problems. Noting that, to address such topic, it is necessary to transfer into a reduced space not only the data but also the constraints and the objective function of the problem. This transference can easily become extremely complex. The idea persecuted in this work concerns making an evolutionary metaheuristic to evolve in a reduced dimension space S', controlling its evolution in the original space S. The transition between S and S' is made with recourse to an autoencoder, applied as a reversible mapping between the two spaces (autoencoders give mappings in both directions between S and S'). This way, the evolutionary metaheuristic can evolve in S', with the assessment of its objective function made in S.

This paper includes the description of the methodology implemented to address optimization in large scale problems, firstly presented in (Costa, 2008) and here designated as "Hybrid". The Hybrid methodology was tested for 9 case studies: seven mathematic optimization functions and two power system problems concerning the coordination of energy production in hydro and wind farms.

The results obtained showed that the Hybrid approach lead to the achievement of better quality solutions for some of the addressed case studies, and these are Sphere functions and the Wind-Hydro coordination problem. The tests conducted with other functions, e.g. Rosenbrock or Rastrigin, did not return a significant gain nor in computational effort or in the quality of the solution achieved. The results and discussion for each case are detailed.

The paper is structured as follows. Section 2 explains the Hybrid procedure implemented. Section 3 provides information on the notation used and on the specifications assumed for all experiments. Sections 4 to 12 present the results and discussion concerning the experiments conducted for the case studies Alpine, Rosenbrock, Griewank, Sphere, Shift Rastrigin, Shift Schwefel, Shift Sphere and Hydro-Wind problem, respectively. The final conclusions are considered in Section 12.

9 2 Hybrid Approach

The Hybrid methodology is next described. Let us consider three major parts: A, B and C.

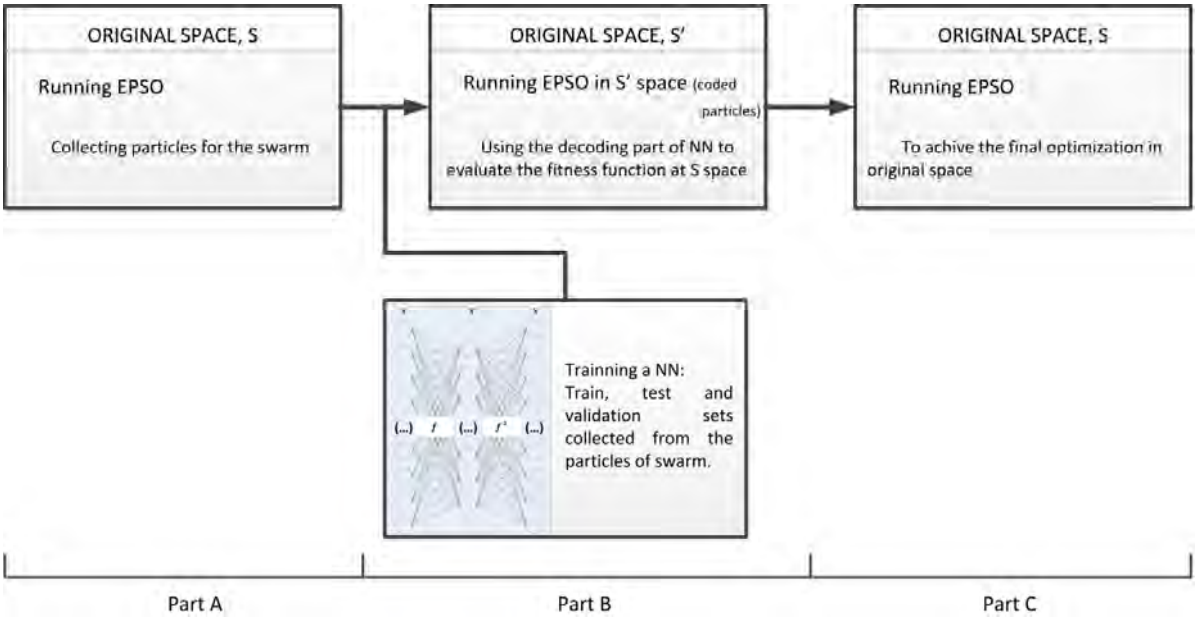


Figure R1/ 1 - Scheme of Hybrid Approach.

Part A

The metaheuristic EPSO is applied within the original space S of the problem. This implementation is made for a specified number of iterations, this number is chosen depending on the problem addressed. The algorithm is allowed to evolve normally and during this period a set of particles is stored, as well as their weights concerning inertia, cooperation, perturbation and memory. The particles stored are to be distinct. The number of particles to store is specified. When the number of particles to store exceeds the maximum number specified, the best ones are kept, and the worst are discarded.

Part B

The set of particles stored is divided in order to construct three datasets: train dataset, test dataset and validation dataset. The construction of these three datasets is made with an alternate and proportional selection of examples from the set of particles.

These datasets are used to train an autoencoder, using a suitable training procedure. Once the autoencoder is trained, the 1st half is used to compress the particles from the original space S to the reduced space S'. The weights of inertia, cooperation, perturbation and memory are kept. The velocity of the particles can be transferred or can be initialized randomly in S'. At this point, the evolutionary metaheuristic EPSO is set to run within space S'. The value of the fitness at each moment is calculated using the 2nd half of the autoencoder to expand and calculate the respective fitness.

Part C

After running a specified number of iterations in S', the swarm is transferred to the original space again, where it can evolve for some more iterations, in order to improve the solution achieved. Next chapters present the results for the two approaches, EPSO and Hybrid, using a tripartite graph, corresponding to the three parts A, B and C. Next graph illustrates an example, where the horizontal axis relates to the number of iterations produced in EPSO and the vertical axis concernes the fitness function value. For Hybrid approach, at the transition from part A to part B the neural network is trained but graph only shows the EPSO iterations.

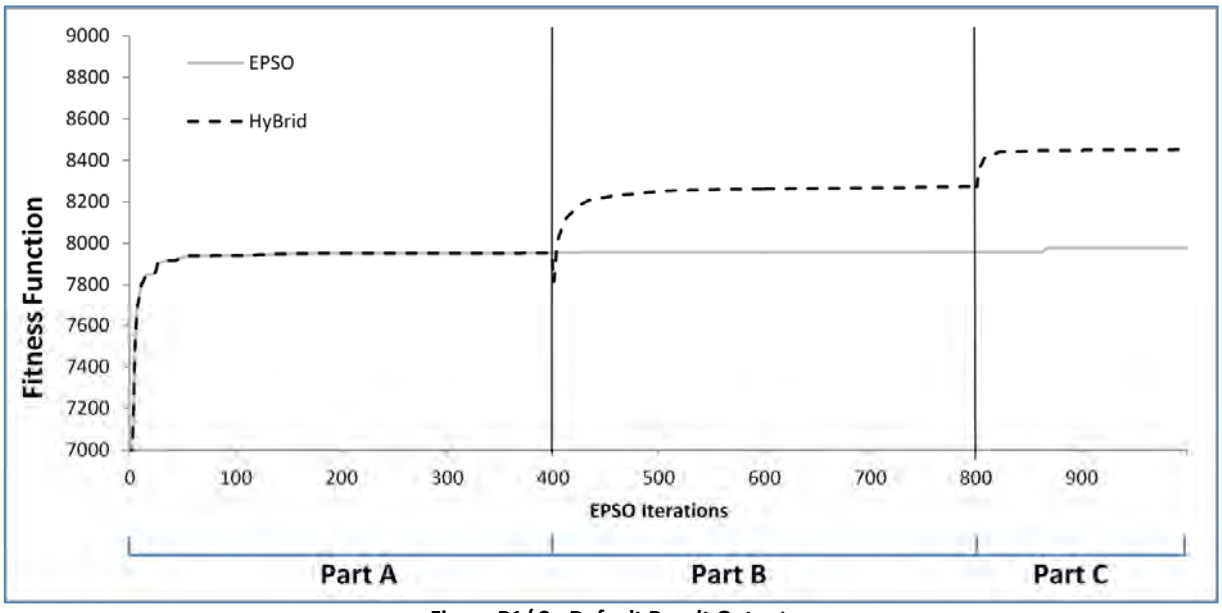


Figure R1/ 2 - Default Result Output.

10 3 General specifications

The original space is denoted with the symbol S and the reduced space with S'. The reference to the dimensions of the spaces used in each experiment is made using squared brackets. For EPSO experiments the dimension of S is mentioned (e.g. EPSO [dim 120]). For Hybrid experiments the three dimensions pursued are included by order: dimension of S, dimension of S' and dimension of S (e.g. Hybrid [dim 120-50-120]).

Autoencoder specifications

For all experiments, autoencoders were trained with a dataset of 1000 examples, a test dataset of 500 examples and a validation dataset of 500 examples. The synaptic weights for the 1st half of the autoencoder were initialized using Principal Components Analysis (PCA), and the synaptic weights for the 2nd half were

initialized with the transposed matrix of the synaptic weights found for the 1st half. The initialization for all synaptic bias was set with zeros. The training includes an adaptive learning rate, as specified in (Hagan, Demuth, & Beale, 1996, pp. 12-12). Also a non-saturation procedure was implemented, allowing the autoencoder to avoid the saturation of the synaptic weights during the training.

Runs

Each experiment refers to the average of 10 runs, each run concerns a distinct initialization of random numbers. Although the initialization of synaptic weights used in all autoencoders is not dependent on randomness, the initialization of swarms and its components is dependent on randomness, alongside with the corresponding evolution in ***S*** and ***S'***.

Results presentation

The results concerning the benchmark optimization functions are presented with several experiments, referring to different dimensions of ***S*** and ***S'***. The results concerning the Hydro-Wind problem are presented with a single experiment each, as the changing of dimensions in these cases represents a different problem.

11 4 Alpine Function

The Alpine function is defined as follows:

$$f(x_1, \cdots, x_D) = \prod_{i=1}^{D} \sin(x_i) \sqrt{\prod_{i=1}^{D} x_i} \qquad \textbf{Eq. R1/(1)}$$

In R^2 and considering a domain $[0,10]^2$, the visualization is provided in the next figure.

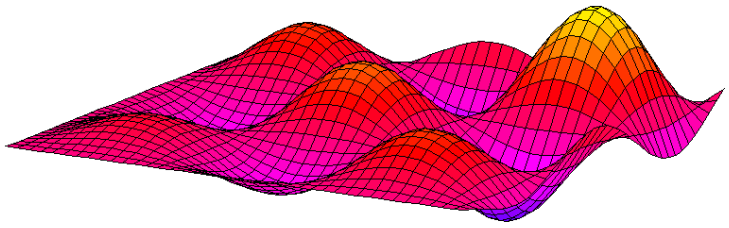


Figure R1/ 3 - Visualization of Alpine function for R^2, from (Clerc, 1998).

Experiment 1 [dim 120 – 50 – 120]

This experiment considers ***S*** to have a dimension of 120, and ***S'*** of 50, for a domain $[0, 10]^{120}$. The autoencoder was trained with backpropagation (PROP) (Rumelhart, Hinton, & Williams, 1986) during 2000 epochs. The 2nd layer included *hyperbolic tangent* activation functions; the 3rd layer included *piecewise-*

linear activation functions. Data were normalized with MinMax Global (Palma & Hora, 2012). Learning rate was initialized with the value 0.5. The parameters found to best perform with EPSO in ***S*** are $\tau_S = 0.4$ and $cp_S = 0.1$. Concerning the EPSO running in ***S'*** the best parameters found were: $\tau_{S'} = 0.4$ and $cp_{S'} = 0.95$. Swarms of size 400 were considered. For the Hybrid approach, the number of iterations used in parts A, B and C were 100, 100 and 100, respectively.

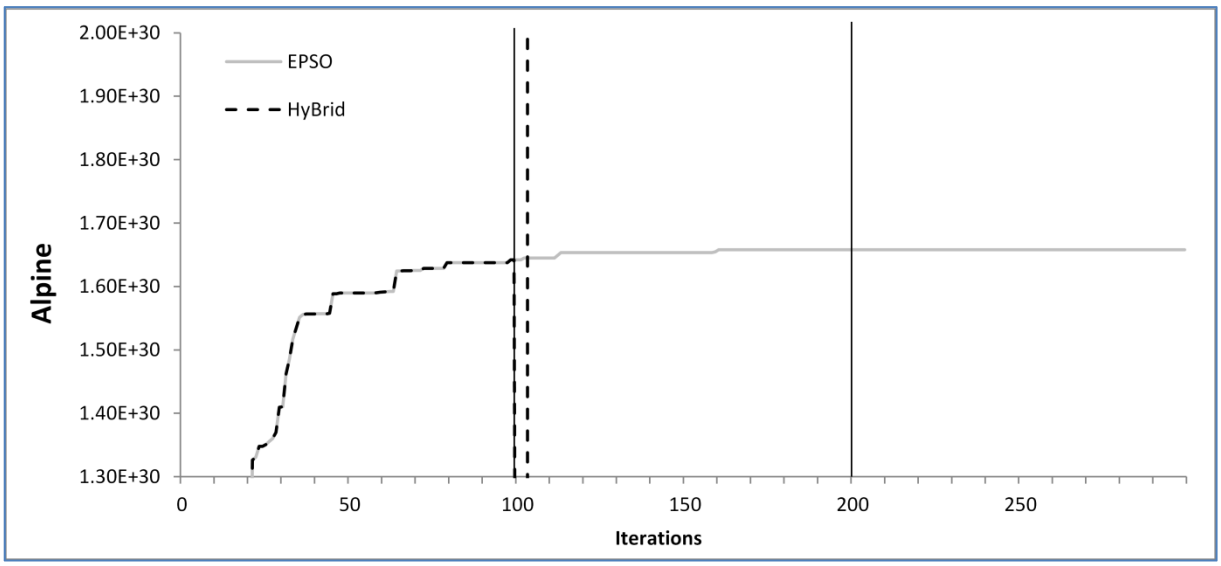


Figure R1/ 4 - Results for Alpine function with Hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

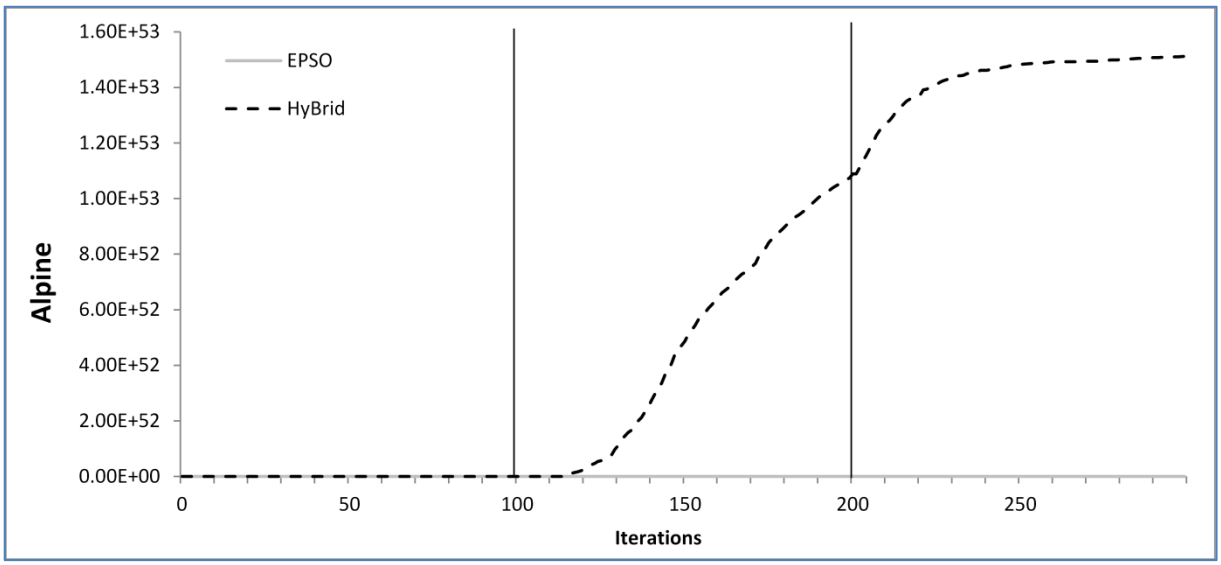


Figure R1/ 5 - Results for Alpine function with Hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

The results obtained are presented in the above figure. For this case, the solution achieved with the hybrid approach are not only higher and better, as they also evidence a magnitude of a higher order (1.51E+53) than the one observed with the classic EPSO (1.65E+30). The above two figures show the Alpine function optimized using EPSO and using the Hybrid approach and the difference between them is on magnitude

scale. The first figure is adjusted for the EPSO fitness solution and the scale of second figure is adjusted for the Hybrid fitness solution.

Experiment 2 [dim 200 – 70 – 200]

The tests conducted concerning this function and this experiment were performed in R^{200} concerning the domain $[0, 10]^{200}$. The same values for the parameters found in experiment 1 are kept, changing only the dimension of the original space (with dimension 200) and of the reduced space (with dimension 70).

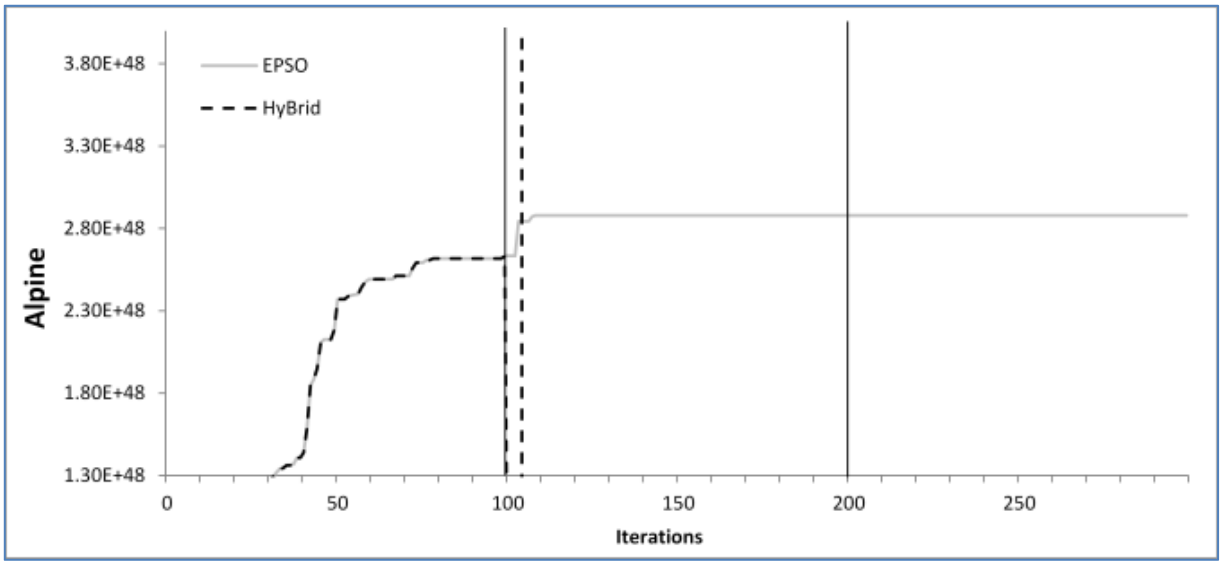


Figure R1/ 6 - Results for Alpine function with Hybrid [dim 200–70–200] and EPSO [dim 200]. Average of 10 runs, scale1.

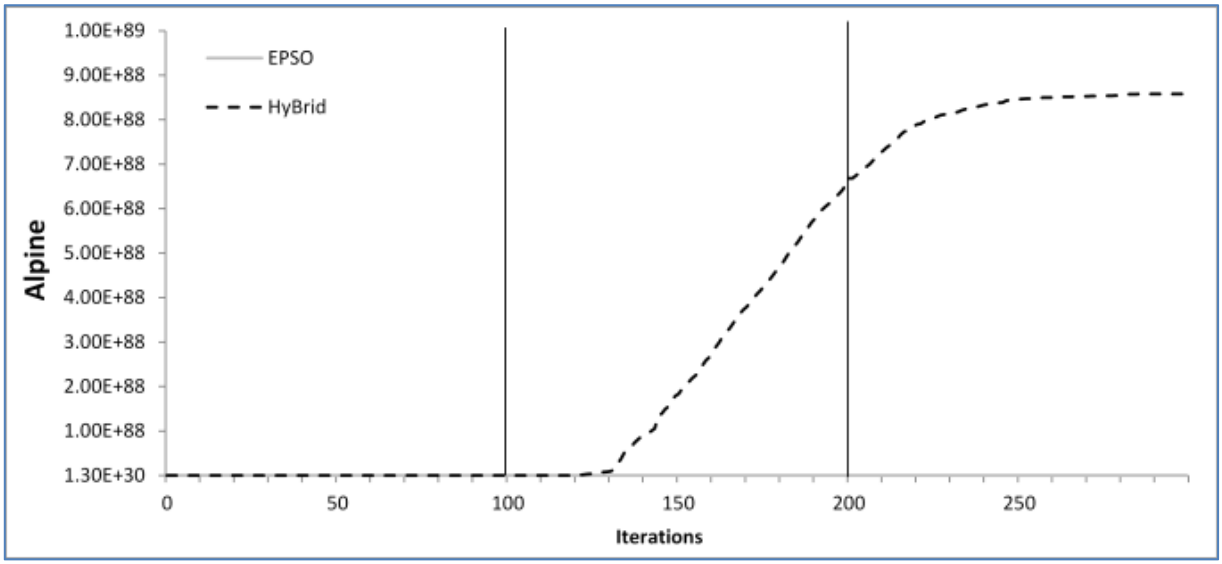


Figure R1/ 7 - Results for Alpine function with Hybrid [dim 200–70–200] and EPSO [dim 200]. Average of 10 runs, scale 2.

The EPSO approach achieved a fitness value of 2.88E+48 after 300 iterations, and the hybrid approach a fitness value of 8.58E+88 for the same number of iterations.

Experiment 3 [dim 200 – 100 – 200]

This experiment is performed in R^{200} concerning the domain $[0, 10]^{200}$. The same values for the parameters found in experiment 1 are kept, changing only the dimension of the original space (with dimension 200) and of the reduced space (with dimension 100).

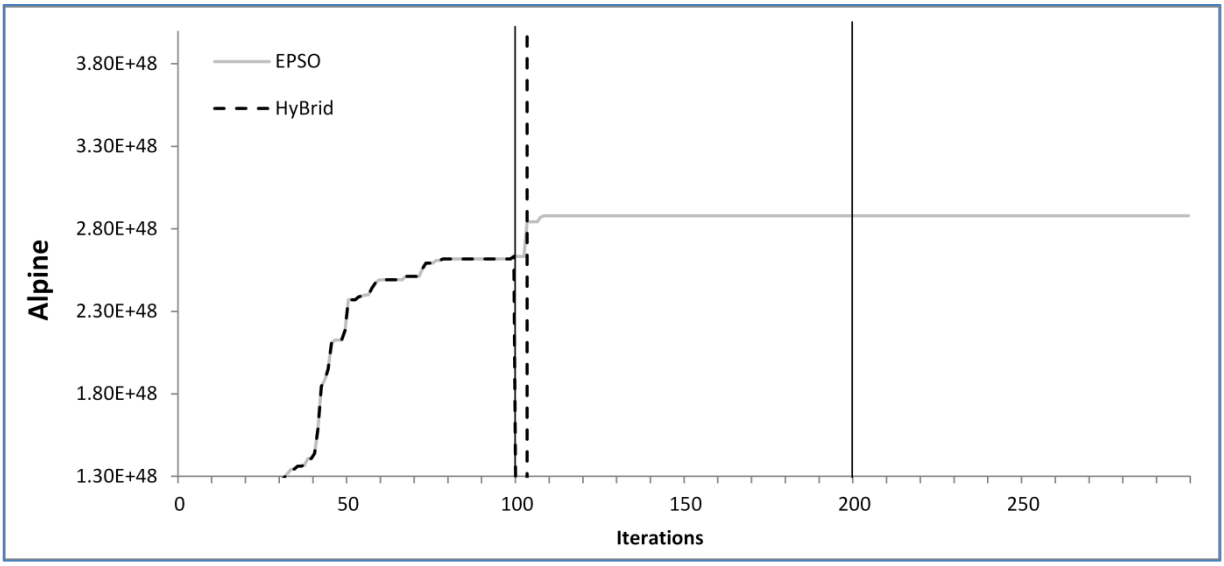


Figure R1/ 8 - Results for Alpine function with Hybrid [dim 200-100-200] and EPSO [dim 200]. Average of 10 runs, scale 1.

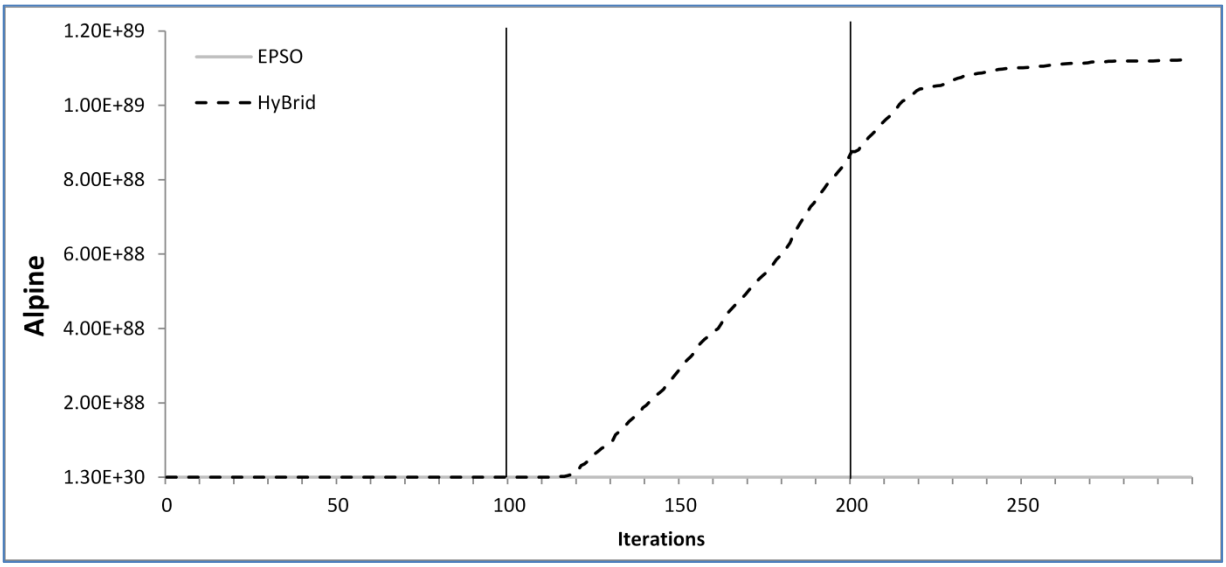


Figure R1/ 9 - Results for Alpine function with Hybrid [dim 200-100-200] and EPSO [dim 200]. Average of 10 runs, scale 2.

The Hybrid approach achieved a final fitness value of 1.12E+89, while the EPSO achieved a maximum fitness of 2.88E+48 after 300 iterations both.

Experiment 4 [dim 300 – 100 – 300]

This experiment considered the original space R^{300} and a domain $[0, 10]^{300}$. All parameter values obtained in experiment 1 were kept. The dimension of the original space was set as 300, and for the small space was set at 100.

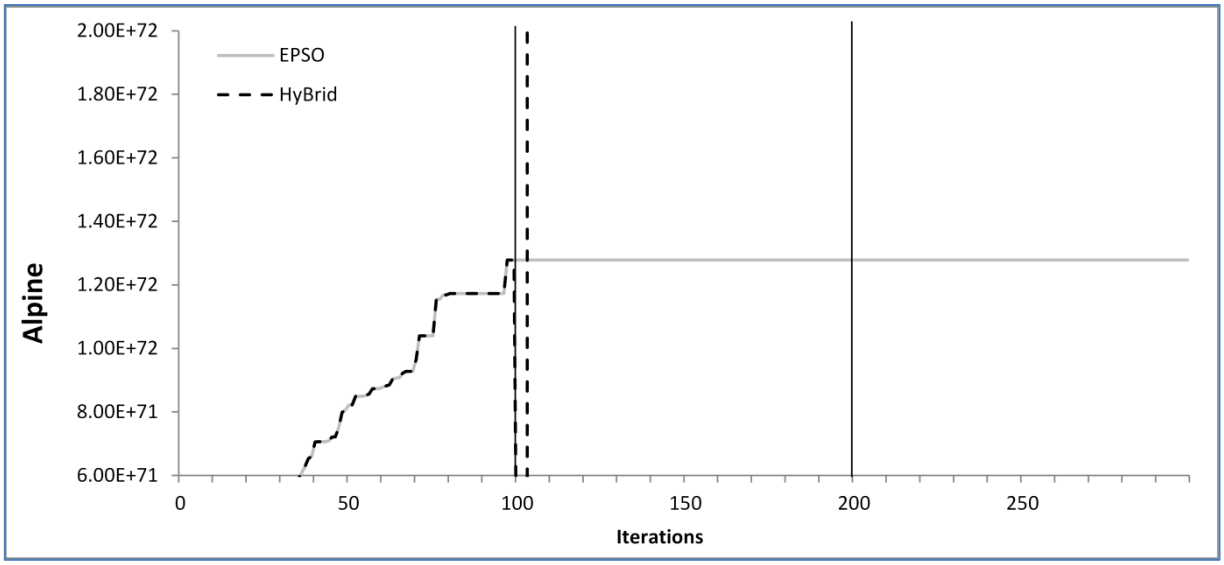


Figure R1/ 10 - Results for Alpine function with Hybrid [dim 300-100-300] and EPSO [dim 300]. Average of 10 runs, scale 1.

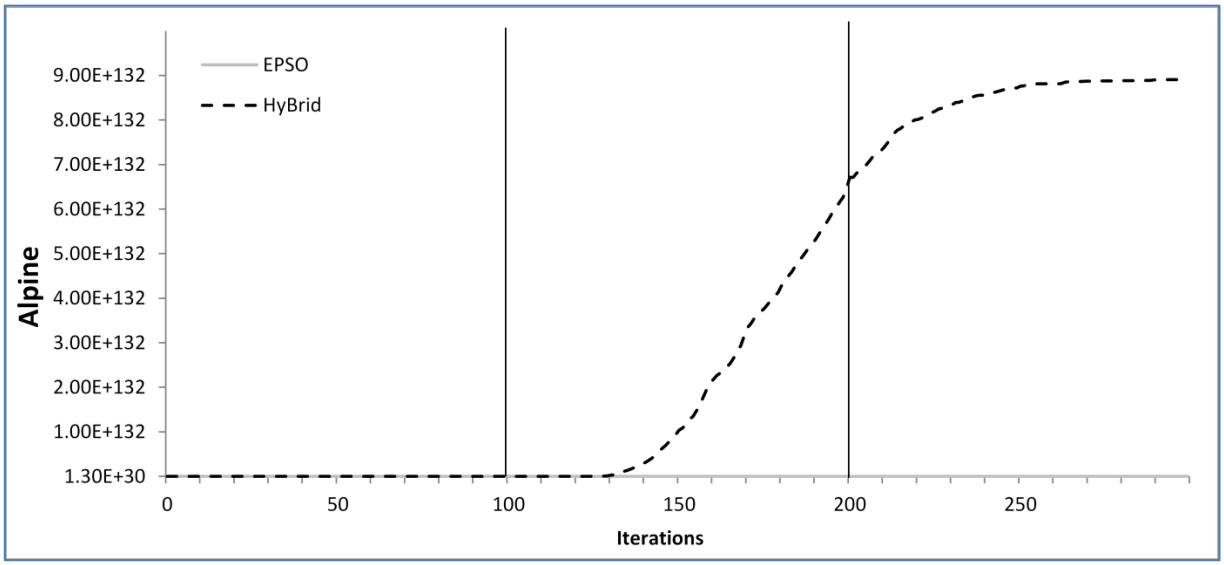


Figure R1/ 11 - Results for Alpine function with Hybrid [dim 300-100-300] and EPSO [dim 300]. Average of 10 runs, scale 2.

Again, the Hybrid approach found fitness values significantly higher than the EPSO approach. The Hybrid approach achieved a final value of 8.91E+132, whilst the EPSO a final fitness value of 1.28E+72, both at the end of 300 iterations.

Experiment 5 [dim 300 – 150 – 300]

Again within an original space R^{300} and with a domain of $[0, 10]^{300}$, this experiment differs from experiment 4 with the specification of a hidden layer with the dimension of 150. All parameter values adopted in experiment 1 were kept.

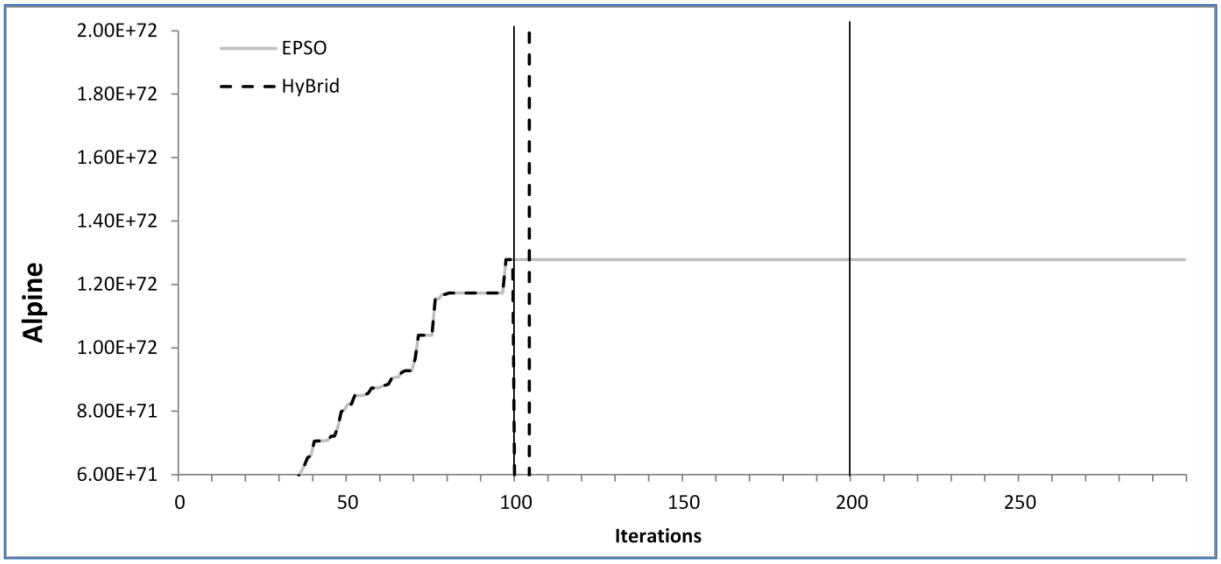


Figure R1/ 12 - Results for Alpine function with Hybrid [dim 300-150-300] and EPSO [dim 300]. Average of 10 runs, scale 1.

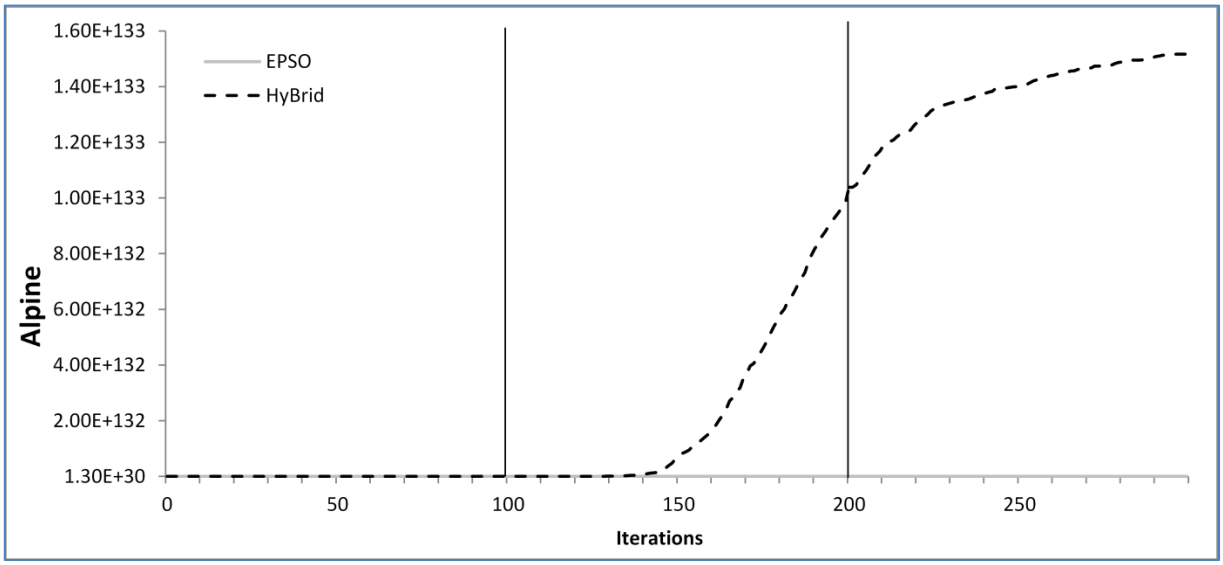


Figure R1/ 13 - Results for Alpine function with Hybrid [dim 300-150-300] and EPSO [dim 300]. Average of 10 runs, scale 2.

The EPSO result obtained in this experiment is similar than the one obtained in experiment 4. The performance of the hybrid approach achieved a final fitness value of 1.52E+133.

12 5 Rosenbrock Function

The Rosenbrock function was firstly proposed in 1960 (Rosenbrock, 1960), and is defined as shown in expression Eq. R1/(2). The visualization of this function in R^2 is provided in Figure R1/ 14 and in Figure R1/ 15.

$$f(x_1, \cdots, x_D) = \sum_{i=1}^{D-1} \left[100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right] \qquad \textbf{Eq. R1/(2)}$$

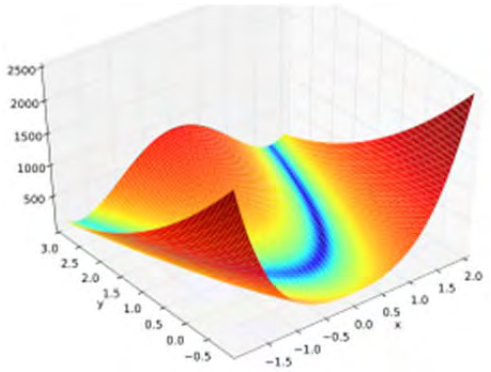


Figure R1/ 14 - Visualization of Rosenbrock function in R^2, from (Wikipedia, 2012b).

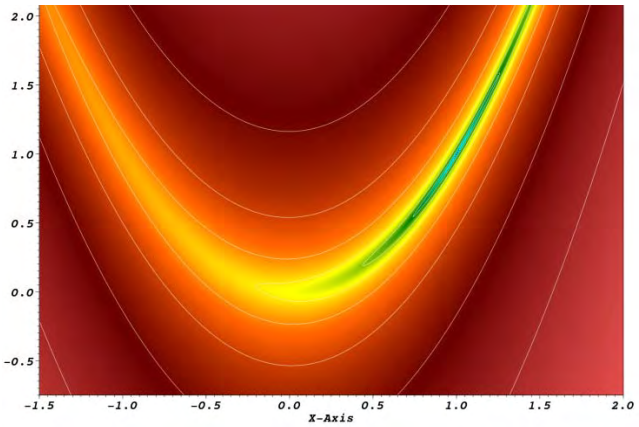


Figure R1/ 15 - Visualization of Rosenbrock function in R^2, from (Richling, 2009).

This function has several local minima and the global one is achieved at $x^* = (1, \cdots, 1)$ with $f(x^*) = 0$ [10, 11, p. 251].

Experiment 1 [dim 120 – 50 – 120]

The dimension of the autoencoder used was specified in $[\dim 120 - 50 - 120]$ and the domain was specified as $[-30, 30]^{120}$. The autoencoder was trained using the ITL plus classic backpropagation algorithm. The activation functions defined for the 2nd and 3rd layers were *hyperbolic tangent* and *piecewise-linear*, respectively. Data were normalized with MinMax by entrance. The initial value of learning rate was specified at 0.5 and the number of epochs applied was 2000. Each swarm included 200 particles. The number of iterations of parts A, B and C were 100, 50 and 150, respectively. The parameters found to best perform with EPSO in ***S*** were $\tau_{\mathbf{S}} = 0.8$ and $cp_{\mathbf{S}} = 0.9$, and in ***S'*** $\tau_{\mathbf{S}'} = 0.6$ and $cp_{\mathbf{S}'} = 0.5$.

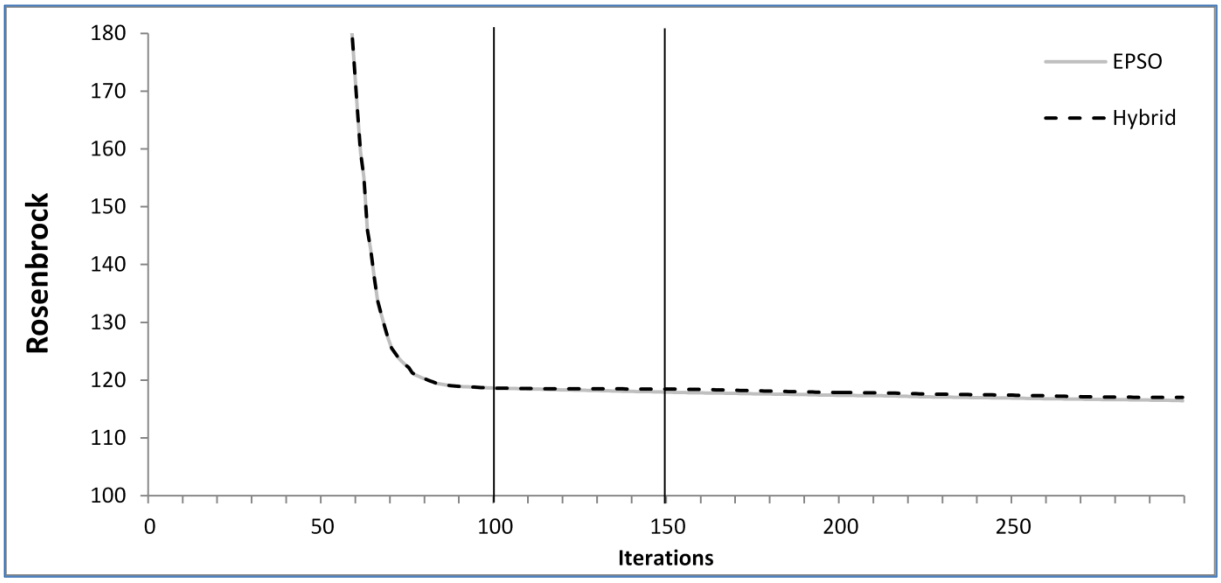


Figure R1/ 16 - Results for Rosenbrock function with Hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

The figure presents the results obtained. For this case, the solution achieved with the hybrid approach (117.008) is equivalent to the EPSO solution (116.461).

Experiment 2 [dim 200 – 70/100 – 200]

Using the values obtained in experiment 1 for all parameters, this experiment induces a change on the dimension of original and reduced spaces. The two reductions were tested: [dim 200-70-200] and [dim 200-100-200].

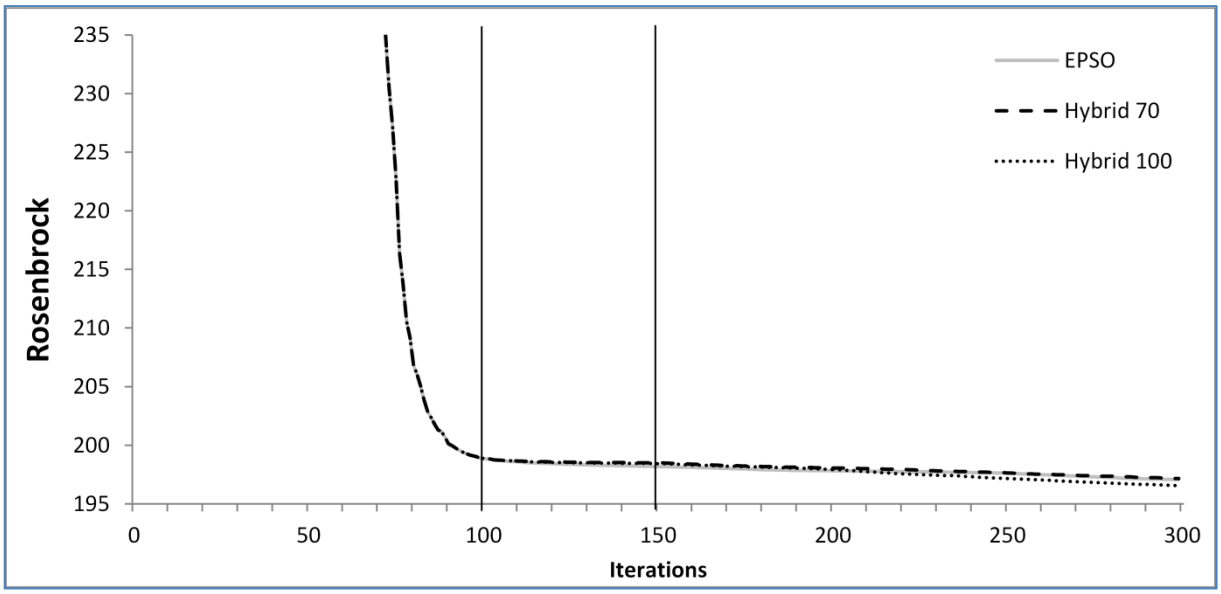


Figure R1/ 17 - Results for Rosenbrock function with Hybrid [dim 200 – 70 – 200], Hybrid [dim 200 – 100-200] and EPSO [dim 200]. Average of 10 runs.

The best fitness value obtained with EPSO was 197.078, which is an equivalent result than the two Hybrid approaches tested: for [dim 200-100-200] the best fitness value obtained was 196.557, and for [dim 200-70-200] the best fitness result was 197.155.

Experiment 3 [dim 300 – 100/150 – 300]

Again, changes on the dimensions of the spaces used are considered. Here the original space is established with dimensionality of 300, and the reduced space was tested with dimensions 100 and 150.

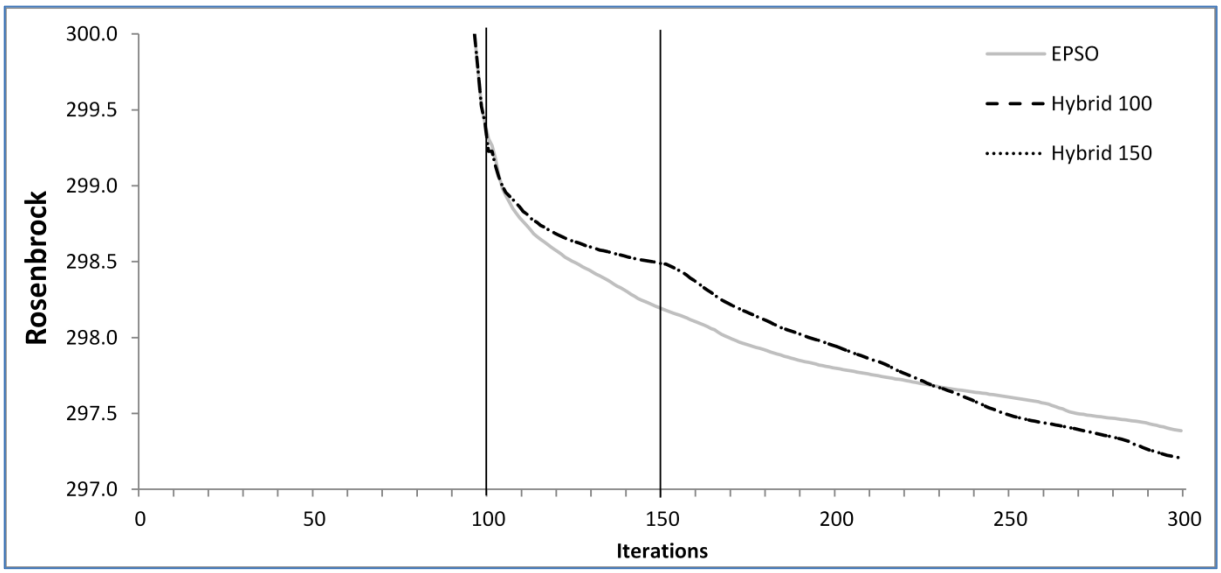


Figure R1/ 18 - Results for Rosenbrock function with Hybrid [dim 300-100-300], Hybrid [dim 300-150-300] and EPSO [dim 300]. Average of 10 runs.

Here, the results obtained with Hybrid approach revealed again equivalent to the EPSO results. The final fitness obtained with EPSO over 300 iterations was 297.387, and with Hybrid [dim 300-100-300] the best fitness was 297.203, and with Hybrid [dim 300-150-300] the best fitness was 297.329.

13 6 Griewank Function

The Griewank function is specified by (Hedar, 2012a; Pan, Suganthan, Tasgetiren, & Liang, 2010; Weisstein, 2012):

$$f(x_1, \cdots, x_D) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) \qquad \textbf{Eq. R1/(3)}$$

The visualization of this function for R^2, concerning the domain $[-30, 30]^2$ is:

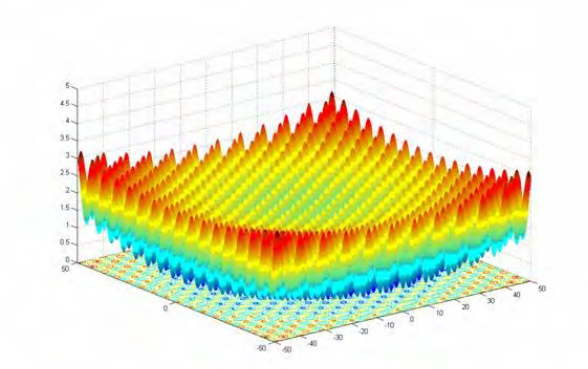


Figure R1/ 19 - Visualization of the Griewank function in R^2, from (Hedar, 2012a).

This function has several local minima and the global one is achieved at $x^* = (0, \cdots, 0)$, $f(x^*) = 0$ (Rao & Savsani, 2012, p. 274).

Experiment 1 [dim 120 – 50 – 120]

This experiment considered an original space dimension of 120, and a reduced space dimension of 50. The domain was specified at $[-30, 30]^{120}$. The autoencoder was trained with maximization of mutual information between the 1st and 2nd layers (1st half), and with backpropagation between 2nd and 3rd layers (2nd half) (Palma & Hora, 2012). The 2nd layer considered the hyperbolic tangent for activation functions, whilst the 3rd layer considered piecewise-linear activation functions. Data were normalized using MinMax by entrance. The initial value of learning rate was specified at 0.5. Each half of the autoencoder was trained with 2000 epochs. The parameters for EPSO in S where tested. A swarm of 400 particles were selected. The number of iterations used were 15 in part A, 50 in part B and 100 in part C. The parameters found to best perform with EPSO for space S were $\tau_S = 0.6$ and $cp_S = 0.95$. For the space S' the parameters found to best perform were $\tau_{S'} = 0.7$ and $cp_{S'} = 0.5$.

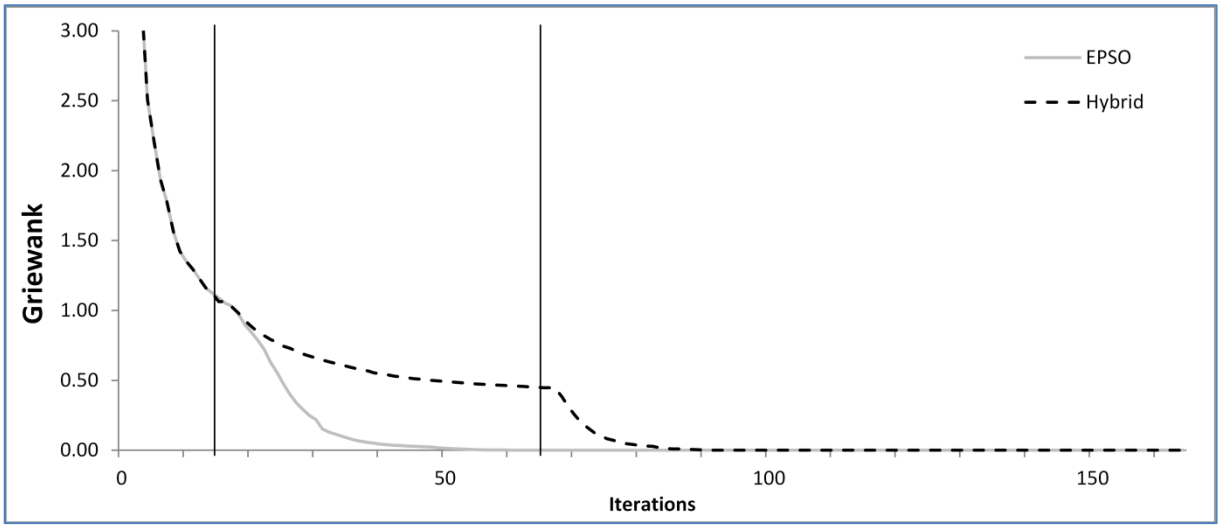


Figure R1/ 20 - Results for Griewank function with Hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

For this experiment the Hybrid [dim 120-50-120] returned a final fitness value of 3.56-12, which is a worst performance than the one observed with EPSO [dim 120], with a final fitness of 5.55E-17.

Experiment 2 [dim 200 – 70/100 – 200]

The same parameters from experiment 1 were adopted. The original space dimension was set at 200. The reduced space dimensions tested were 70 and 100.

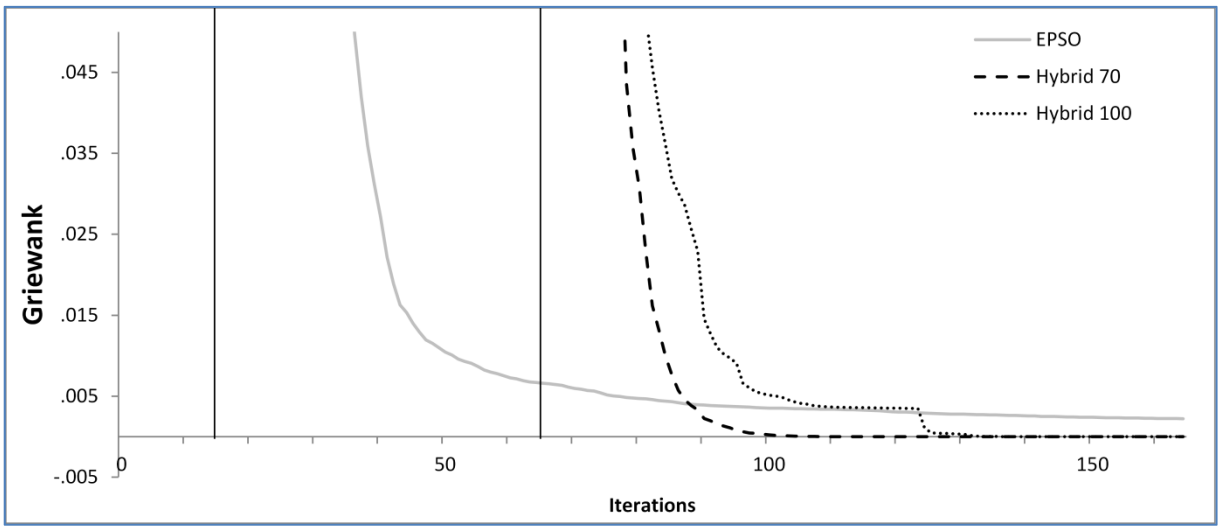


Figure R1/ 21 - Results for Griewank function with Hybrid [dim 200 – 70 – 200], Hybrid [dim 200-100-200] and EPSO [dim 200]. Average of 10 runs.

The EPSO metaheuristic returned a fitness value of 2.22E-03 after 165 iterations. This result was worse than the results obtained with the two experiments applying the hybrid approach. With the Hybrid [dim 200 – 70 – 200] the fitness achieved after 165 iterations was 6.63E-12, whilst with Hybrid [dim 200 – 100 – 200] the fitness obtained was 1.12E-09.

Experiment 3 [dim 300 – 100/150 – 300]

For this experiment the original space was set at 300, and the reduced spaces tested were 100 and 150.

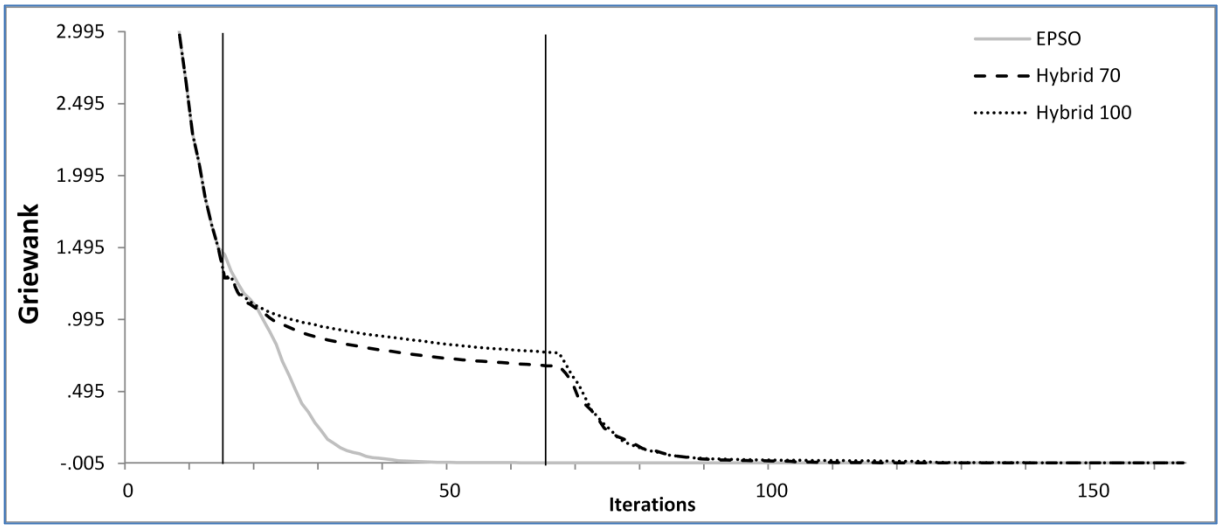


Figure R1/ 22 - Results for Griewank function with Hybrid [dim 300 – 100 – 300], Hybrid [dim 300-150 – 300] and EPSO [dim 300]. Average of 10 runs.

For this experiment the EPSO and Hybrid approaches present, at the end of the 165 iterations, equivalent solutions.

14 7 Sphere Function

The Sphere function is specified by:

$$f(x_1, \cdots, x_D) = \sum_{i=1}^{D} {x_i}^2 \qquad \textbf{Eq. R1/(4)}$$

The visualization of this function for R^2, concerning the domain $[-30, 30]^2$ is:

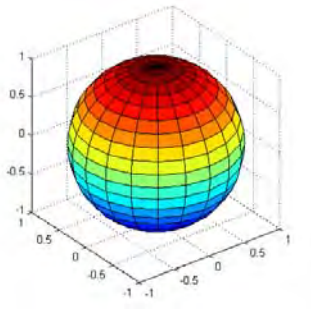


Figure R1/ 23 - Visualization of the Sphere function in R^2 from (MathWorks, 2012).

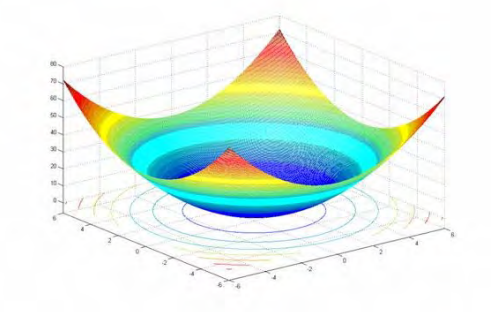


Figure R1/ 24 - Visualization of the Sphere function in R2 from (Hedar, 2012c).

This function has no local minimum except the global one. The global minima is achieved at $x^* = (0, \cdots, 0)$, $f(x^*) = 0$ (Rao & Savsani, 2012, p. 235).

Experiment 1 [dim 120 – 50 – 120]

The dimension of the autoencoder was specified in $[\dim 120 - 50 - 120]$ and the domain was specified as $[-30, 30]^{120}$. For the best result obtained, in addition to the general parameters, the autoencoder was trained using the ITL plus Prop approach. The activation functions for the second and third layer was the hyperbolic tangent. The normalization was made using MinMax Global algorithm. The initial value of learning rate was specified at 0.5 and the number of epochs applied was 2000 for first part using ITL training and more 2000 for the second half using Prop training. The parameters for EPSO in space S were tested. A swarm of 200 particles were selected. The number of iterations used were 100 in part A, 100 in part B and 100 in part C. The parameters found to best perform with EPSO for space S were $\tau_S = 0.7$ and $cp_S = 0.9$. For the space S' the parameters found to best perform were $\tau_{S'} = 0.8$ and $cp_{S'} = 0.5$.

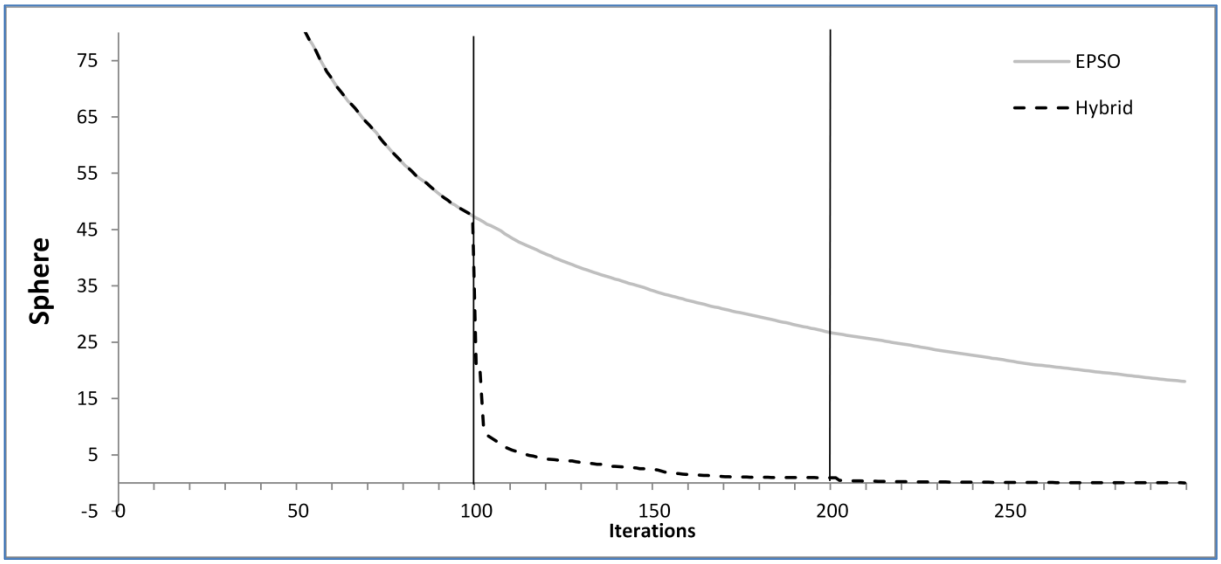


Figure R1/ 25 - Results for Sphere function with hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

For this experiment the hybrid approach returned a significantly better performance with a final fitness value of 0.0396142 in comparison with EPSO approach which achieved a final fitness value of 18.0566, both with 100 iterations.

Experiment 2 [dim 200 – 70/100 – 200]

Using the same parameters, changing the dimensions for the original and reduced spaces. The original space dimension was set in 200. The reduced space dimensions tested were 70 and 100.

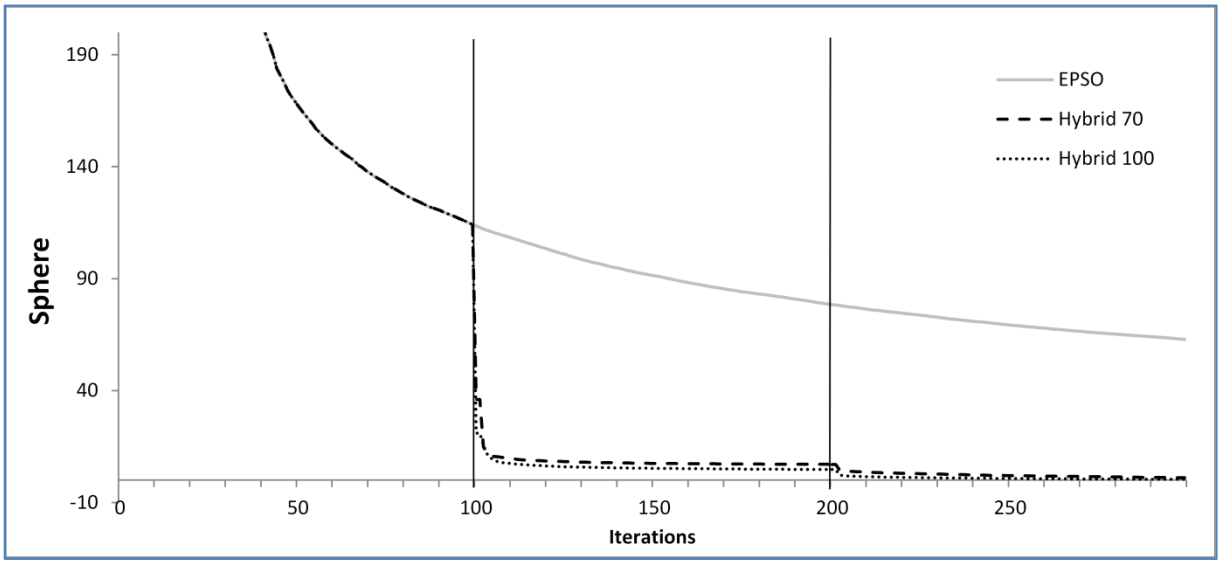


Figure R1/ 26 - Results for Sphere function with hybrid [dim 200 – 70 - 200], hybrid [dim 200-100-200] and EPSO [dim 200]. Average of 10 runs.

It is possible to observe that both Hybrid approaches returned best performance and significantly better final results than the EPSO approach with a final fitness value of 62.8726, during the 300 iterations. The differences between the hybrid approaches were not significant, both returned accurate results: hybrid [dim 200-70-200] with a final fitness of 1.07285 and hybrid [dim 200 – 100 – 200] with a final fitness of 0.360871.

Experiment 3 [dim 300 – 100/150 – 300]

For this experiment, an original space of dimension 300 was considered. Two reduced space dimensions were tested: 100 and 150.

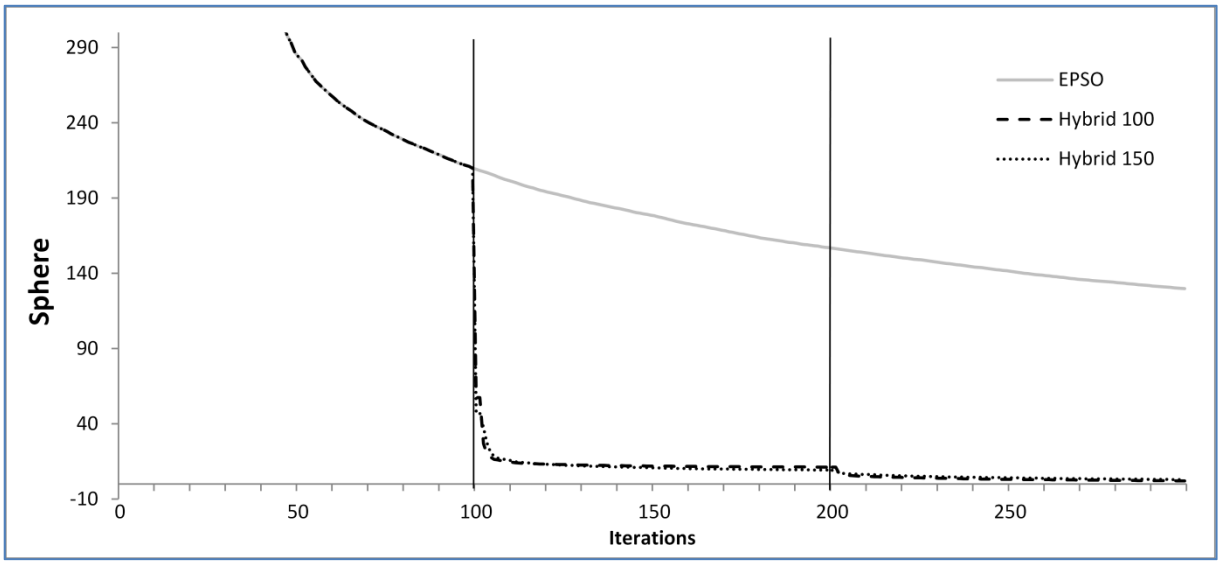


Figure R1/ 27 - Results for Sohere function with Hybrid [dim 300-100-300], Hybrid [dim 300-150-300] and EPSO [dim 300]. Average of 10 runs.

The results obtained suggest the adequacy of the hybrid approach, which returned better results than EPSO for both dimensions tested: hybrid [dim 300-100-300] with a final fitness value of 2.13499 and [dim 300-150-300] with a final fitness value of 2.93594. The EPSO returned a final fitness value of 129.813.

15 8 Shift Rastrigin Function

The Shift Rastrigin function is specified as follows:

$$f(x_1, \cdots, x_D) = A \cdot n + \sum_{i=1}^{D} [x_i^2 - A \cdot \cos(2\pi x_i)] + f_{bias} \qquad \textbf{Eq. R1/(5)}$$

Where $A = 10$ and $f_{bias} = -330$.

The visualization of this function for R^2 is provided in Figure R1/ 28, concerning the domain $[-5.12, 5.12]^2$ (Pan et al., 2010).

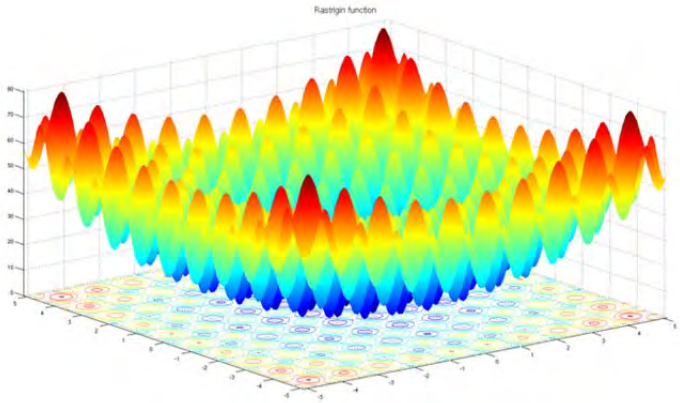


Figure R1/ 28 - Visualization of Shift Rastrigin function in R^2, from (Wikipedia, 2012a).

This function has several local minima, the global minimum is achieved at $x^* = (0, \cdots, 0), f(x^*) = -330$ (Rao & Savsani, 2012, p. 266).

Experiment 1 [dim 120 – 50 – 120]

The tests conducted concerning this function and this experiment were performed concerning the domain $[-5.12, 5.12]^{120}$. In that case, the dimension of the autoencoder used was specified as $[\dim 120 - 50 - 120]$. The autoencoder was trained with PROP during 2000 epochs. The adopted activation function for the 2nd layer was hyperbolic tangent, and for 3rd layer piecewise-linear. Data were normalized with MinMax Global (Palma & Hora, 2012). The adaptive learning rate started at 0.5. The best parameters found for EPSO in ***S*** were: $\tau_S = 0.9$ and $cp_S = 0.9$, and for EPSO in ***S'***: $\tau_{S'} = 0.8$ and $cp_{S'} = 0.4$. Each swarm included 400

particles. The number of iterations for parts A, B and C were 100, 30, 100, respectively. The results obtained for this experiment are present in Figure R1/ 29.

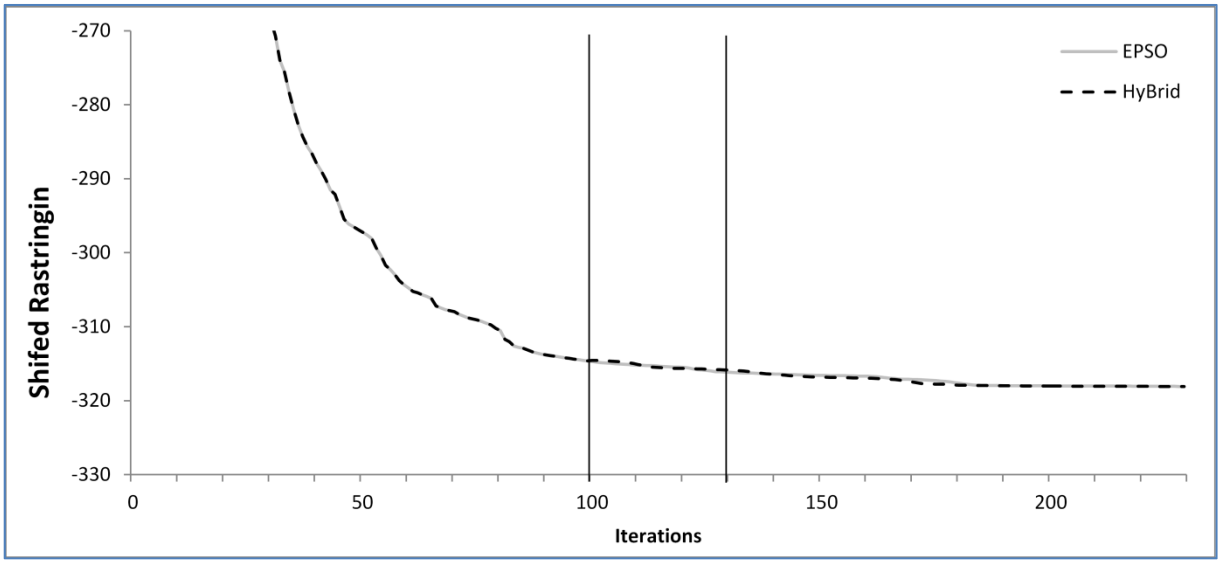


Figure R1/ 29 - Results for Shift Rastrigin function with Hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

For this case the EPSO achieved an equivalent solution to the Hybrid approach. The final fitness value obtained with Hybrid [dim 120-50-120] was -318.09, whilst with EPSO [dim 120] for the same number of iterations the fitness achieved was -318.08.

Experiment 2 [dim 200 – 70/100 – 200]

Using the same parameters, changing the dimensions for the original and reduced spaces. The original space dimension was set in 200. The reduced space dimensions tested were 70 and 100.

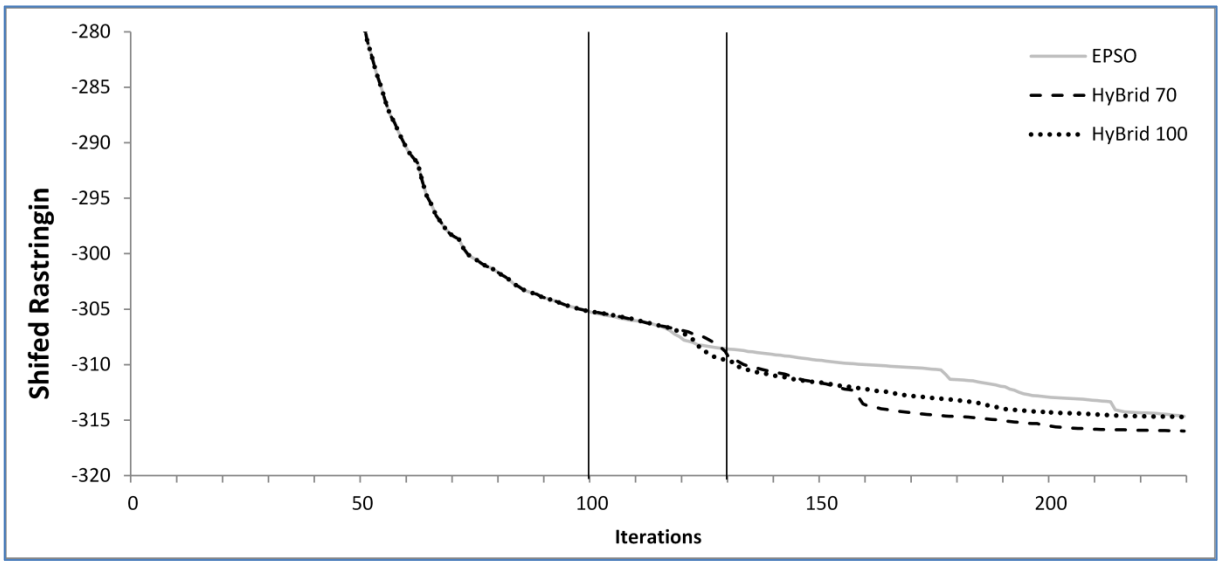


Figure R1/ 30 - Results for Shift Rastrigin function with hybrid [dim 200 – 70 - 200], hybrid [dim 200-100-200] and EPSO [dim 200]. Average of 10 runs.

Here, each swarm includes 200 particles. It is possible to observe that Hybrid 70 approach returned best performance (final fitness values of -315.983) than the EPSO approach with a final fitness value of -314.651, during the 230 iterations. The Hybrid 100 approach reveals an equivalent performance to EPSO with a final fitness value of and -314.712.

Experiment 3 [dim 300 – 100/150 – 300]

For this experiment, an original space of dimension 300 was considered. Two reduced space dimensions were tested: 100 and 150. Here, each swarm includes 200 particles.

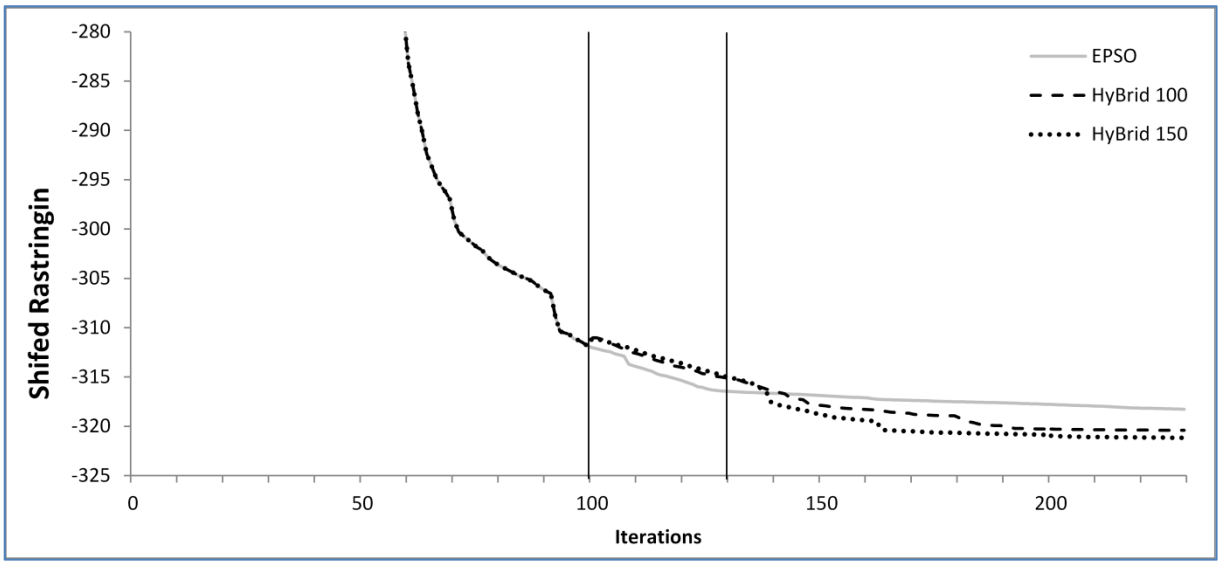


Figure R1/ 31 - Results for Shift Rastrigin function with Hybrid [dim 300-100-300], Hybrid [dim 300-150-300] and EPSO [dim 300]. Average of 10 runs.

The results obtained suggest the adequacy of the hybrid approach, which returned better results than EPSO for both dimensions tested: hybrid [dim 300-100-300] with a final fitness value of -320.408 and [dim 300-150-300] with a final fitness value of -321.161. The EPSO approach returned a final fitness value of -318.264.

16 9 Shift Schwefel Function

The Shift Schwefel function (*Problem 1.2*) is specified as follows:

$$f(x_1, \cdots, x_D) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i \right)^2 + f_{bias} \qquad \textbf{Eq. R1/(6)}$$

Where $f_{bias} = -450$. The visualization of this function in R^2 is provided in Figure R1/ 32 and in Figure R1/ 33.

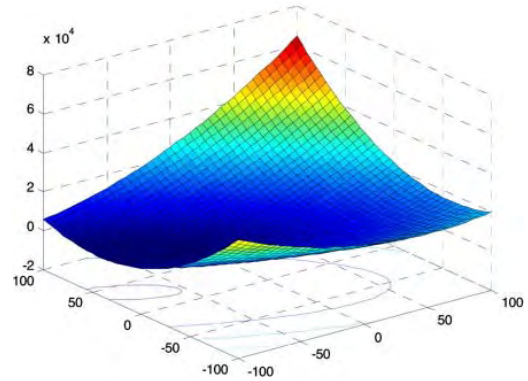


Figure R1/ 32 - Visualization of Shift Schwefel function in R^2, from (Pan et al., 2010).

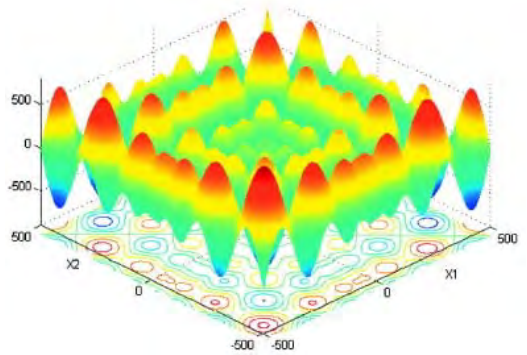


Figure R1/ 33 - Visualization of Schwefel function in R^2, from (Hedar, 2012b).

This function has several local minima. The global minima is achieved at $x^* = (0, \cdots, 0)$, $f(x^*) = -450$ (Rao & Savsani, 2012, p. 243).

Experiment 1 [120 – 50 – 120]

The dimension of the autoencoder was specified in $[\dim 120 - 50 - 120]$ and the domain was specified as $[-30, 30]^{120}$. The autoencoder was trained using the ITL (Cauchy-Schwartz) plus Prop approach. The adopted activation functions for the 2nd and 3rd layers were hyperbolic tangent and piecewise-linear, respectively. Data were normalized with MinMax Entrance (Palma & Hora, 2012). The adaptive learning rate started with the value 0.5. The number of epochs applied was 2000 for first part using ITL training and more 2000 for the second half using Prop training. The parameters for EPSO in space S where tested. A swarm of 400 particles were selected. The number of iterations used were 10 in part A, 10 in part B and 60 in part C. The parameters found to best perform with EPSO for space S were $\tau_S = 0.8$ and $cp_S = 0.9$. For the space S' the parameters found to best perform were $\tau_{S'} = 0.1$ and $cp_{S'} = 0.9$.

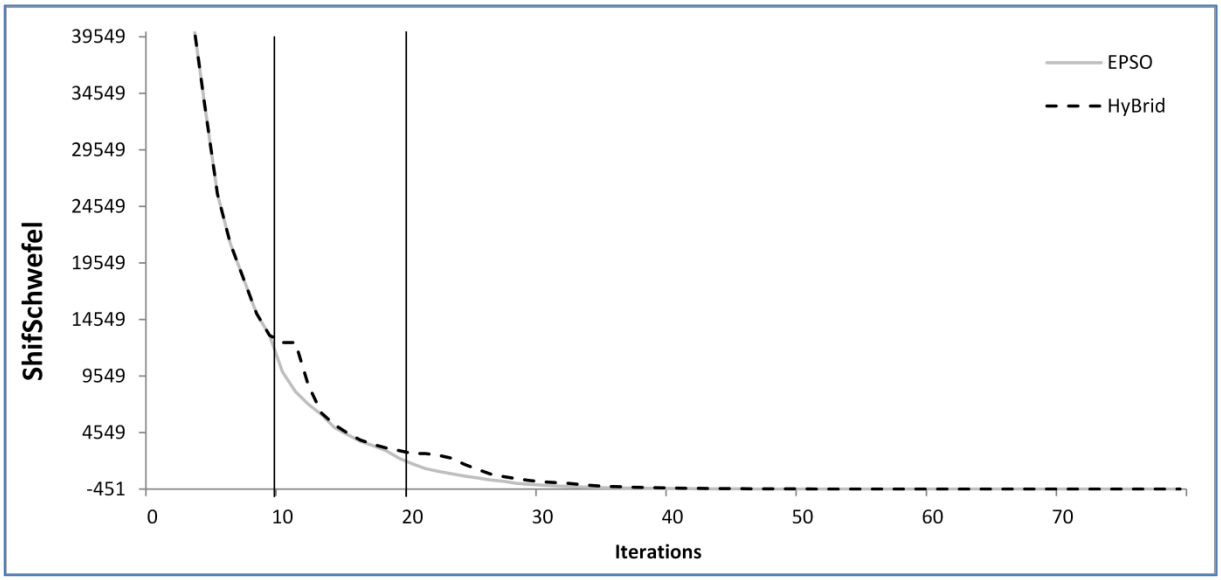


Figure R1/ 34 - Results for Shift Schwefel function with Hybrid [dim 120 – 50 – 120] and EPSO [dim 120]. Average of 10 runs.

Experiment 2 [dim 200 – 70/100 – 200]

Using the same parameters of experiment 1 for an original space dimension of 200. The reduced space dimensions tested were 70 and 100.

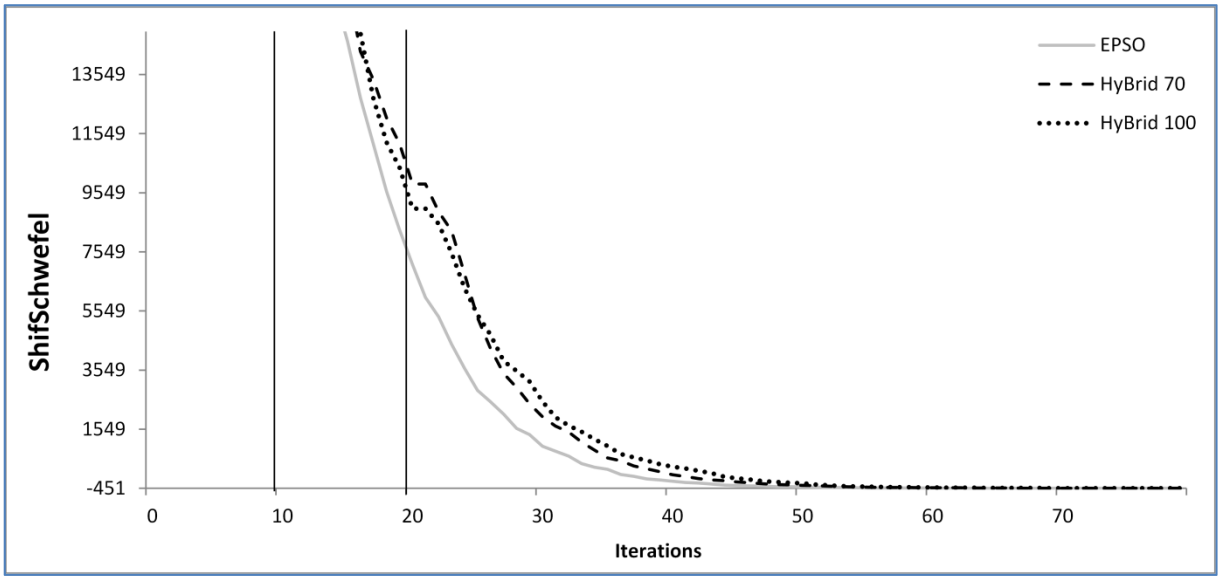


Figure R1/ 35 - Results for Shift Schwefel function with Hybrid [dim 200 – 70/100 – 200] and EPSO [dim 200]. Average of 10 runs.

The EPSO result revealed slightly better than both hybrid approaches: hybrid [dim 200 – 70 – 200] with a final fitness value of -449.329, the hybrid [dim 200-100-200] with -449.154 and EPSO reach the optimum of -449.778.

Experiment 3 [dim 300 – 100/150 – 300]

For this experiment the original space dimension was set at 300. The reduced space dimensions were set at 100 and 150.

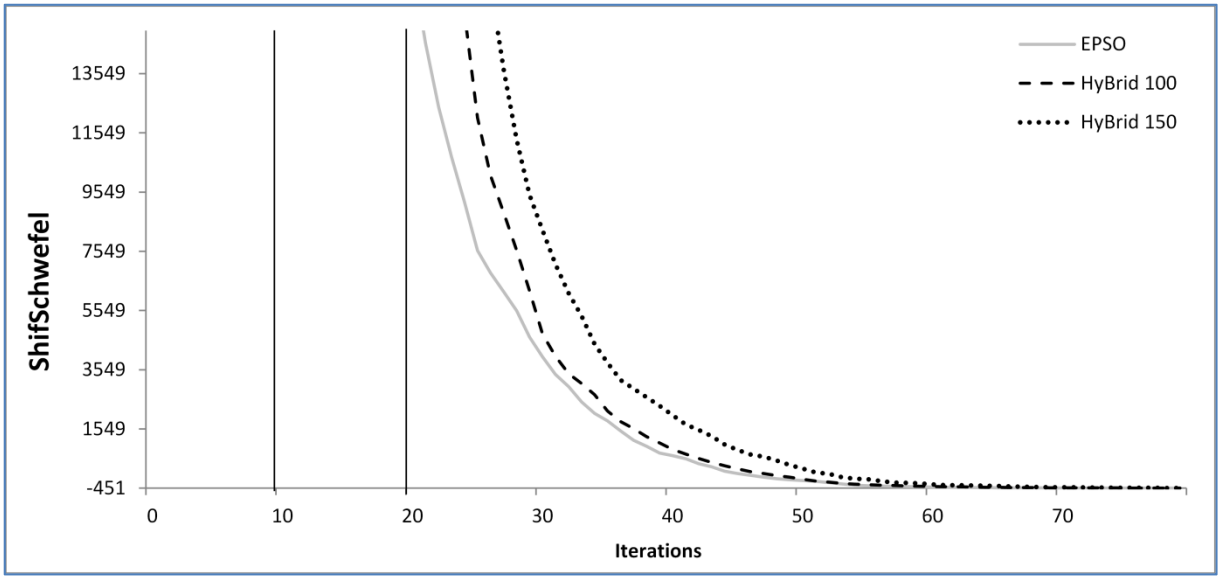


Figure R1/ 36 - Results for Shift Schwefel function with Hybrid [dim 300 – 100/150 – 300] and EPSO [dim 300]. Average of 10 runs.

The EPSO result revealed better than both hybrid approaches: hybrid [dim 300 – 100 – 300] with a final fitness value of -447.226, the hybrid [dim 300 – 150 – 300] with -444.968 and EPSO with a fitness value of -448.684.

17 10 Shift Sphere Function

The Shift Sphere function is specified as follows:

$$f(x_1, \cdots, x_D) = \sum_{i=1}^{D} {x_i}^2 + f_{bias} \qquad \textbf{Eq. R1/(7)}$$

Where $f_{bias} = -450$.

The visualization of this function for R^2, concerning the domain $[-100,\ 100]^2$ is further provided:

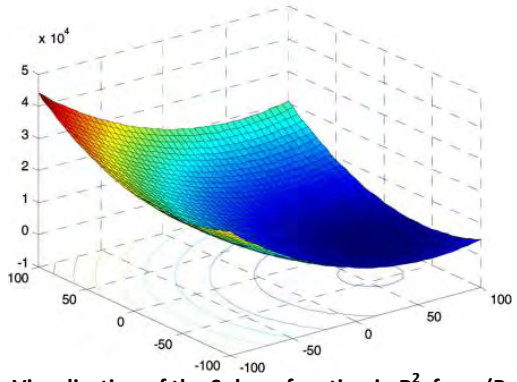


Figure R1/ 37 - Visualization of the Sphere function in R^2, from (Pan et al., 2010).

Experiment 1 [dim 120 – 50 – 120]

The dimension of the autoencoder was specified in $[\dim 120 - 50 - 120]$ and the domain was specified as $[-100, 100]^{120}$. For the best result obtained, in addition to the general parameters, the autoencoder was trained using the ITL plus Prop approach. The activation functions are hyperbolic tangent for the second and the third layer. The normalization was made using MinMax Global algorithm. The initial value of learning rate was specified at 0.5 and the number of epochs applied was 2000 for first part using ITL training and more 2000 for the second half using Prop training. The parameters for EPSO in space S where tested. A swarm of 400 particles were selected. The number of iterations used were 100 in part A, 100 in part B and 100 in part C. The parameters found to best perform with EPSO for space S were $\tau_S = 0.7$ and $cp_S = 0.9$. For the space S' the parameters found to best perform were $\tau_{S'} = 0.8$ and $cp_{S'} = 0.5$.

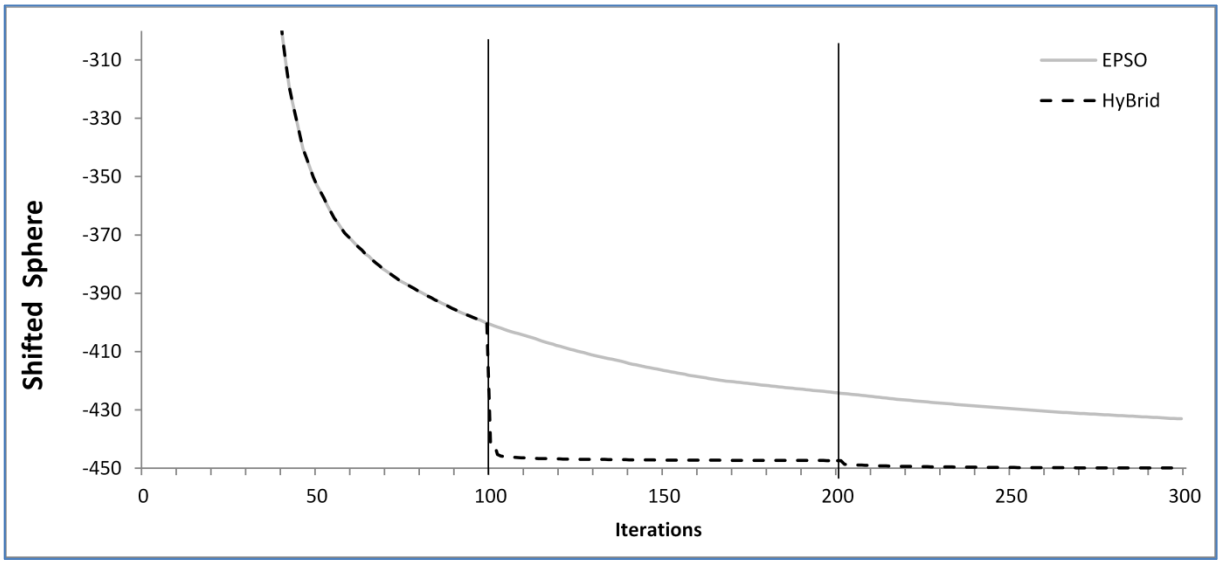


Figure R1/ 38 - Results for Shift Sphere function [dim 120 – 50 – 120]. Average of 10 runs.

Experiment 2 [dim 200 – 70/100 – 200]

Using the same parameters of experiment 1 for an original space dimension of 200. The reduced space dimensions tested were 70 and 100.

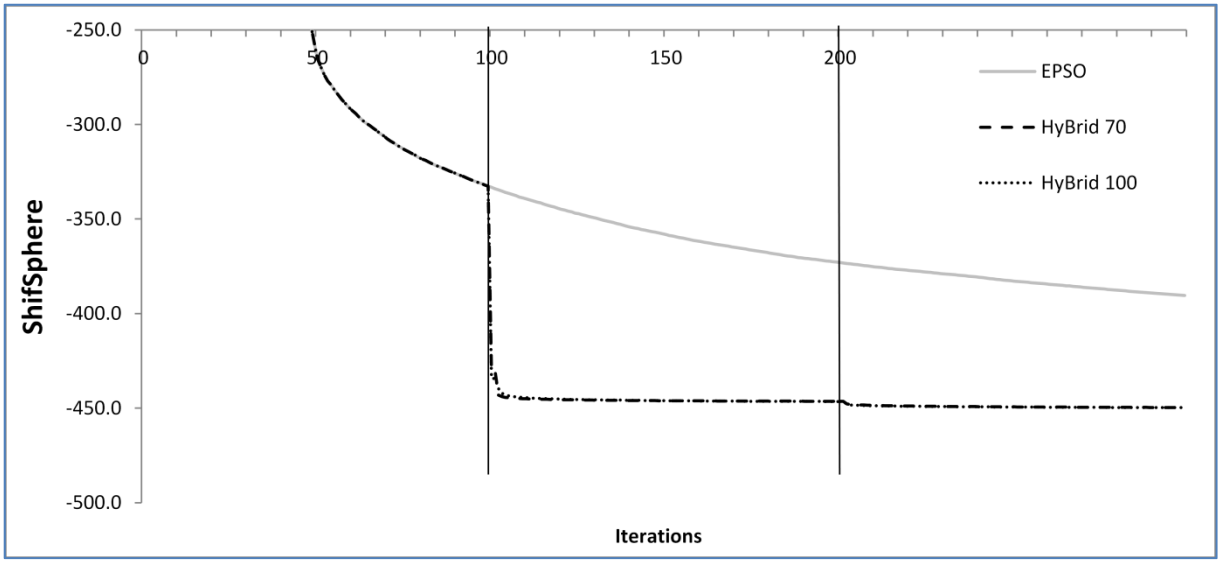


Figure R1/ 39 - Results for Shift Sphere function with Hybrid [dim 200-70-200], Hybrid [dim 200-100-200] and EPSO [dim 200]. Average of 10 runs.

Both hybrid approaches showed a better performance than EPSO: hybrid [dim 200 – 70 – 200] with a final fitness value of -449.691, the hybrid [dim 200 – 100 – 200] with -449.746 and EPSO with -390.464.

Experiment 3 [dim 300 – 100/150 – 300]

For this experiment the original space dimension was set at 300. The reduced space dimensions were set at 100 and 150.

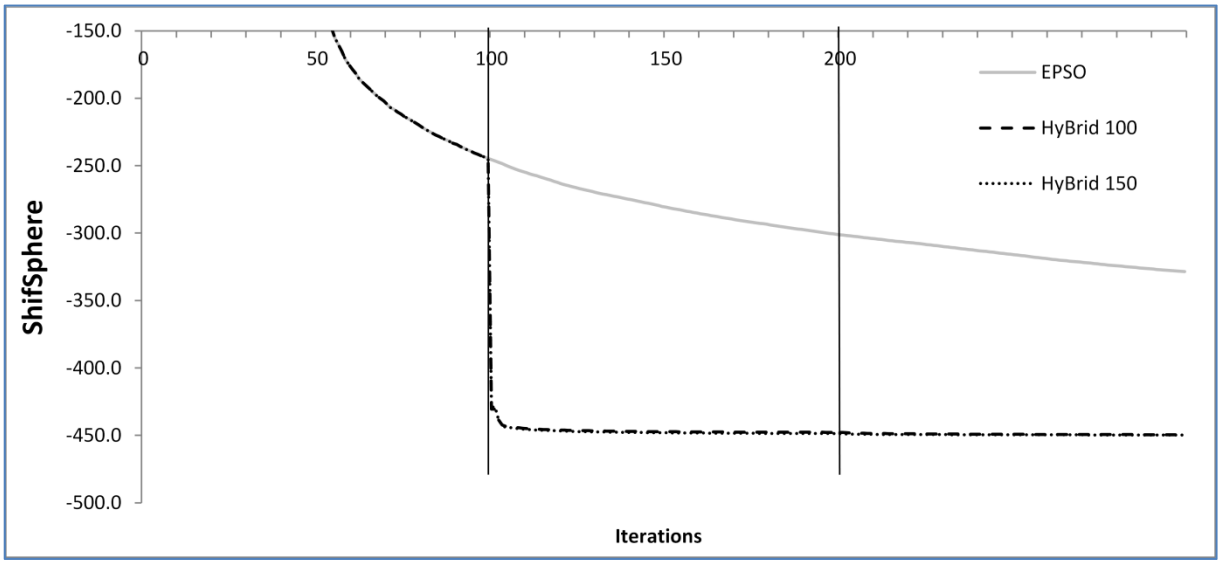


Figure R1/ 40 - Results for Shift Sphere function with Hybrid [dim 300-100-300], Hybrid [dim 300-150-300] and EPSO [dim 300]. Average of 10 runs.

Both hybrid approaches returned better results than EPSO. Hybrid [dim 300 – 100 – 300] with a final fitness value of -449.547, Hybrid [dim 300 – 150 – 300] with -449.754, and EPSO with -328.48.

18 11 Hydro-Wind problem

The Hydro-Wind / Hydrothermal / Hydroelectric coordination problem aims at maximizing the joint profit or minimize the joint cost of a power system composed by several hydro and wind farms. The maximization / minimization is made by changing the water volumes to be pumped and released, given a set of specifications defining each scenario. These specifications include the wind forecast or the water inflow to the system. Due to the high complexity this problem can achieve, the operation planning is normally made with multiple approaches for different horizons of analysis (Lyra, Tavares, & Soares, 1984): from the short term (days) to the long term (years).

This problem has been widely studied: in (Chang, Chen, Fong, & Luh, 1990) this problem is addressed with a differential dynamic programming algorithm, decomposition and coordination techniques are considered in (Soares, Lyra, & Tavares, 1980) and a genetic algorithm approach was proposed in (Zoumas, Bakirtzis, Theocharis, & Petridis, 2004). An insightful review concerning applications and methods for this problem is provided in (Labadie, 2004).

The problem formulation followed in this work is similar to the one proposed in (Soares & Carneiro, 1991), which considers the optimal operation in a deterministic context, meaning that future inflows (of water, of

wind energy) are considered as an assumption – although these values are now actually known, the objective is to analyze the system response to a specific situation (*a posteriori* analysis).

The problem presented in this work considers differentiation concerning peak and off-peak periods: the power demand suffers high variations concerning day and night periods. Moreover, these demands also vary from season to season (the season variation is not contemplated in this formulation, since we are mostly devoted to solve a short period problem). Other energy sources, such as nuclear and fossil fuel plants, are inefficient in generating power for short periods of increased demand (US DIBRPRO, 2005). On the other hand, hydroelectric generators can be started and stopped almost instantly, making the energy produced in hydro farms timely responsive to peak demands. Water can be stored in reservoirs during off-peak periods, and used to produce energy during the peak periods.

This section is organized as follows. Sub-section 11.1 contains a detailed Hydro-Wind problem mathematical formulation. Sub-sections 11.2 and 11.3 include the results and discussion on the application of Hybrid and EPSO to the Hydro-Wind case studies of 8 and 12 reservoirs, respectively.

18.1 11.1 Problem formulation

The general definition of potential energy (E, in J) is given by equation Eq. R1/(8).

$$E = m \cdot g \cdot h \qquad \textbf{Eq. R1/(8)}$$

E: potential energy (J);

m: mass (kg);

g: gravity acceleration (m/s^2);

h: height (m);

Considering a generic time moment Δt divided in both terms of equation Eq. R1/(8), and knowing that ${}^{m}/_{\Delta t} = \rho \cdot \dot{q}$, the equation Eq. R1/(9) can be formulated.

$$\frac{E}{\Delta t} = \frac{m}{\Delta t} \cdot g \cdot h = \rho \cdot g \cdot h \cdot \dot{q} \qquad \textbf{Eq. R1/(9)}$$

Δt:time moment (s);

ρ: water density (kg/m^3);

$\dot{q}$: flow rate (m^3/s);

Equation Eq. R1/(9) can be reorganized as specified in equation Eq. R1/(10).

$$E = \rho \cdot g \cdot h \cdot \dot{q} \cdot \Delta t \qquad \textbf{Eq. R1/(10)}$$

A Hydro farm produces energy by employing the principle described in equation Eq. R1/(10) using turbines, either to pump or to turbine water. This process is associated with energy losses, from where the consideration of an efficiency term η, which is directly multiplied to equation Eq. R1/(10), leading to equation Eq. R1/(11).

$$E = \eta \cdot \rho \cdot g \cdot h \cdot \dot{q} \cdot \Delta t \qquad \textbf{Eq. R1/(11)}$$

The electric energy of hydro origin generated in moment t by reservoir n is described by equation Eq. R1/(12).

$$H_{n,j}^{t} = K_{n,j}[h_n(x_n^t) - h_n(q_n^t + z_n^t)] \cdot |q_n^t| \qquad \textbf{Eq. R1/(12)}$$

where:

- $H_{n,j}^{t}$ - the energy generated by the n^{th} reservoir in period t if $j = turbine$, or the energy consumed if $j = pump$ (in Wh);
- N – the number of hydro power plants included in the system;
- $K_{n,j}$ - a specified constant for each reservoir, which considers the gravitational acceleration (g), the efficiency of the turbine (η) and the water density (ρ): $K_{n,j} = \eta \cdot \rho \cdot g$. This constant takes different values for pumping ($j = pump$) and generation ($j = turbine$) modes, concerning the different efficiencies involved;
- x_n^t - the volume stored in the n^{th} reservoir at the beginning of the period t (in m^3);
- q_n^t - volume of water transferred between the n^{th} and the immediately downstream reservoirs, at moment t: assumes negative values for volumes pumped into the reservoir, and positive values for volumes released out of the reservoir (in m^3);
- z_n^t - the volume of water spilled during the period t (in m^3);
- $h_n(\cdot)$ – function returning the estimation of the water head (height) given a water volume, for the n^{th}reservoir (in m).

The available water volume for each reservoir is calculated for each period considering the variables associated to the reservoir, such as the natural affluences, the volume of water pumped or used in generation, the volume of water spilled and finally the already existing water volume, all of them in the previous period of time, and also considering the variables associated to the operation of the upstream reservoirs such as the quantity of water that was used for generation and now haves to be accommodated in the downstream reservoirs and also the water volume spilled from the upstream reservoirs. So in the Hydro-Wind coordination model, the procedure above is mathematically represented as dictated in equation Eq. R1/(13).

$$x_n^{t+1} = x_n^t + y_n^t + \sum_{k \in \Omega} [q_k^t + z_k^t] - q_n^t - z_n^t \qquad \textbf{Eq. R1/(13)}$$

where:

- y_n^t – volume of water entering the n^{th} reservoir concerning natural river inflow (in m^3);
- Ω – set of hydro reservoirs immediately upstream of the n^{th} reservoir;

Under this case study, the EPSO algorithm is applied to optimize a particle $\boldsymbol{q}$, which includes the volumes q_n^t for each reservoir under each temporal moment, as specified in Eq. R1/(14), where N refers to the total number of reservoirs, and T to the total number of temporal moments.

$$\boldsymbol{q} = [q_1^1, q_2^1, \cdots, q_N^1, \cdots, q_1^T, q_2^T, \cdots, q_N^T] \qquad \textbf{Eq. R1/(14)}$$

Constraints ensuring reservoirs' capacities

Assuming that the system always starts with water volumes x_n^t respecting the corresponding reservoir minimum and maximum capacity limits m_n and M_n, respectively, the model must ensure these limits are satisfied in further temporal moments. Therefore, the volume of the n^{th} reservoir at moment $t+1$ must respect:

$$m_n < x_n^{t+1} < M_n \qquad \textbf{Eq. R1/(15)}$$

Combining equation Eq. R1/(13) and Eq. R1/(15), one can obtain the dynamic constraints specified in Eq. R1/(16) and Eq. R1/(17), which are applied to each position of $\boldsymbol{q}$.

$$x_n^t + y_n^t + \sum_{k \epsilon \Omega} [q_k^t + z_k^t] - z_n^t - M_n < q_n^t \qquad \textbf{Eq. R1/(16)}$$

$$x_n^t + y_n^t + \sum_{k \epsilon \Omega} [q_k^t + z_k^t] - z_n^t - m_n > q_n^t \qquad \textbf{Eq. R1/(17)}$$

Constraints ensuring turbines' capacities

For each reservoir, the specifications on the turbines installed were considered, which allowed the estimation of maximum and minimum volumes they are able to release or pump. These constraints are considered in the model as represented in Eq. R1/(18).

$$q_n^{min} < q_n^t < q_n^{max} \qquad \textbf{Eq. R1/(18)}$$

Constraints ensuring the available water to pump

When the decision to pump water is made, the maximum value of volume to pump must also be restricted to the available volume in the immediately downstream reservoir (IDR). This constraint is only meaningful to the pumping case since when releasing water, even if the IDR exceeds its maximum capacity, it would spill out the overflow. Accordingly, the maximum volume of water to pump into the n^{th} reservoir, γ_n^t, is defined in Eq. R1/(19), and constrains q_n^t as defined in Eq. R1/(20).

$$\gamma_n^t = x_{IDR}^t - m_{IDR} \qquad \textbf{Eq. R1/(19)}$$

$$\gamma_n^t < q_n^t \qquad \textbf{Eq. R1/(20)}$$

The energy generated at wind farms per period is estimated as w^t by an external forecasting procedure – and taken as data in this example of coordination planning. Its value per period is derived from the wind series and each wind farm production characteristic, which can be modeled separately from the optimization procedure. In fact, as there are no "reservoirs for wind", the generation forecast is a direct function of the wind forecast. An auxiliary vector $\boldsymbol{w}$ is considered, where each element w_n^t refers to the available wind energy for the n^{th}reservoir at moment t, and this vector is updated as further described.

Value of energy produced by water released

When q_n^t takes a positive value, meaning the decision of releasing water was made, the corresponding energy is calculated following equation Eq. R1/(12). The value associated to this energy is further calculated by considering the corresponding price, depending if the period type is peak or off-peak, as detailed in Eq. R1/(21).

$$V_{n,A}^t = \begin{cases} H_{n,turb}^t \cdot P_{A,1} & \text{,if under peak period} \\ H_{n,turb}^t \cdot P_{A,0} & \text{,if under off-peak period} \end{cases} \qquad \textbf{Eq. R1/(21)}$$

Value of energy consumed to pump water

When q_n^t takes a negative value, indicating the decision of pumping, the energy necessary to pump is calculated using equation Eq. R1/(12). Two possibilities may happen when the decision of pumping water is made: there is enough wind energy available w_n^t, to pump the water, or there is not. When there is enough wind energy available (i.e. $w_n^t \geq H_{n,pump}^t$), w_n^t is used. In this case, the value of the wind energy spent to pump is calculated by Eq. R1/(22).

$$V_{n,B}^t = \begin{cases} H_{n,pump}^t \cdot P_{B,1} & \text{,if under peak period} \\ H_{n,pump}^t \cdot P_{B,0} & \text{,if under off-peak period} \end{cases} \qquad \textbf{Eq. R1/(22)}$$

The second possibility is that there is not enough wind energy available (i.e. $w_n^t < H_{n,pump}^t$). In this scenario, the model spends all the available energy from wind farms and buys the remainder necessary energy from grid (i.e. $G_p = H_{n,pump}^t - w_n^t$). In this situation, the equation Eq. R1/(22) is used to calculate the value of the energy consumed from wind farms, and equation Eq. R1/(23) is considered to calculate the value of the energy bought from grid to pump.

$$V_{n,C}^t = \begin{cases} (H_{n,pump}^t - w_n^t) \cdot P_{C,1} & \text{,if under peak period} \\ (H_{n,pump}^t - w_n^t) \cdot P_{C,0} & \text{,if under off-peak period} \end{cases} \qquad \textbf{Eq. R1/(23)}$$

For both cases, the wind energy available is updated, to provide accurate energy assessment of all reservoirs under the same temporal moment.

Value of wind energy

When all reservoirs are assessed for a specified temporal moment, the model will calculate the monetary value of the available wind energy at that moment, if any is available, which is considered to be sold to the grid. This value is estimated as defined in Eq. R1/(24).

$$V_{n,D}^{t} = \begin{cases} w_n^t \cdot P_{D,1} & \text{,if under peak period} \\ w_n^t \cdot P_{D,0} & \text{,if under off-peak period} \end{cases} \qquad \textbf{Eq. R1/(24)}$$

System's revenue

The revenue obtained with each reservoir is defined in Eq. R1/(25):

$$R_n^t = \begin{cases} V_{n,A}^t + V_{n,B}^t - V_{n,B}^t - V_{n,C}^t & \text{,if } n \neq N \\ V_{n,A}^t + V_{n,B}^t - V_{n,B}^t - V_{n,C}^t + V_{n,D}^t & \text{,if } n = N \end{cases} \qquad \textbf{Eq. R1/(25)}$$

Finally, once all reservoirs are assessed for all temporal moments, the profit obtained with the entire system is calculated as defined in Eq. R1/(26).

$$Profit = \sum_{t=1}^{T} \sum_{n=1}^{N} R_n^t \qquad \textbf{Eq. R1/(26)}$$

18.2 11.2 Hydro-Wind problem with 8 Reservoirs

This case study concerns to the wind hydro coordination problem with 8 reservoirs. Next the problem parameters are presented. The next figure shows the structure of the 8 reservoirs:

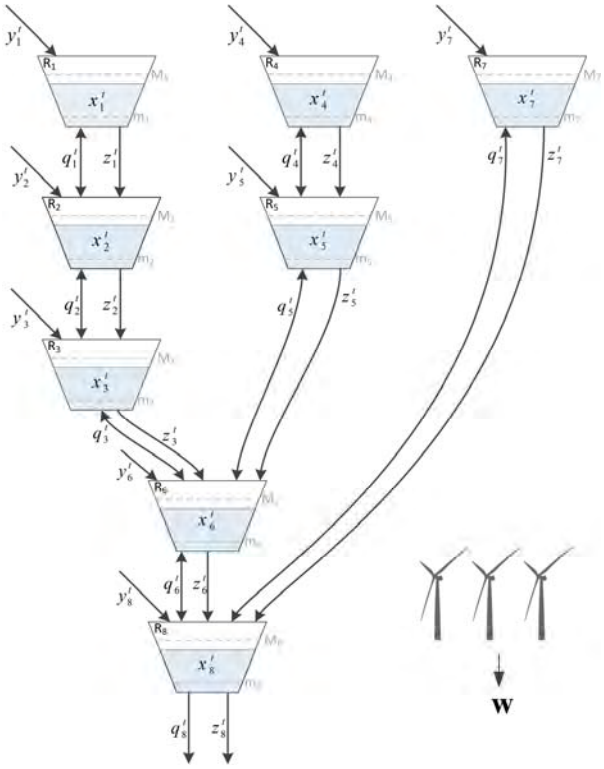


Figure R1/ 41 - Simplified scheme of wind - hydro farms with 8 reservoirs.

Experiment

For this case study, the dimension of S is 96, and the dimension of S' is 50. The autoencoder was trained with PROP during 2000 epochs. The activation functions for the 2nd and 3rd layers were hyperbolic tangent and piecewise-linear, respectively. Data were normalized with MinMax Entrance. The adaptive learning rate was set to start at 0.5. The parameters found to best perform with EPSO in $\boldsymbol{S}$ were $\tau_{\mathbf{S}} = 0.9$ and $cp_{\mathbf{S}} = 0.7$. For the space S' the parameters found to best perform were $\tau_{S'} = 0.8$ and $cp_{S'} = 0.1$. The parameters for EPSO in S were tested. A swarm of 50 particles were selected. The number of iterations used were 400 in part A, 400 in part B and 200 in part C.

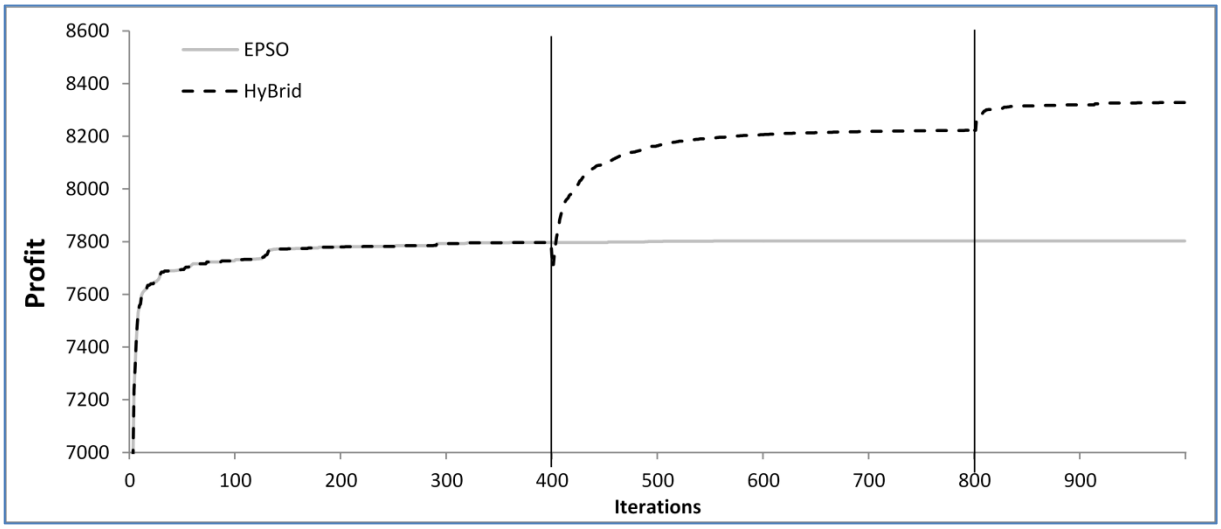


Figure R1/ 42 - Results for 8Reservoirs problem with Hybrid [dim 96 – 50 – 96] and EPSO [dim 96]. Average of 10 runs.

18.3 11.3 Hydro-Wind problem with 12 Reservoirs

This case study concerns to the wind hydro coordination problem with 12 reservoirs. Next figure shows the structure of the 12 reservoirs.

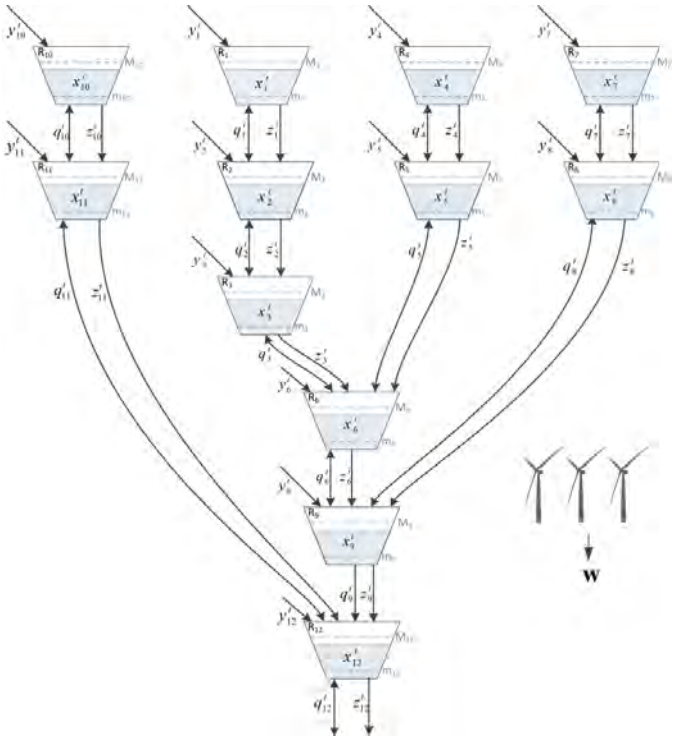


Figure R1/ 43 - Simplified scheme of wind - hydro farms with 12 reservoirs.

Experiment

In this case study we have only one experiment on dimensions that corresponds to the number of input problem variables (number fixed). The dimension of the autoencoder was specified in $[146 - 50 - 146]$.

For the best result obtained, in addition to the general parameters, the autoencoder was trained using the classic backpropagation. The activation functions are hyperbolic tangent for the second layer and piecewise-linear for the third layer. The normalization was made using Global MinMax algorithm. The initial value of learning rate was specified at 0.5 and the number of epochs applied was 2000. The parameters for EPSO in space S were tested. A swarm of 50 particles were selected. The number of iterations used were 400 in part A, 400 in part B and 200 in part C. The parameters found to best perform with EPSO for space S were $\tau_S = 0.9$ and $cp_S = 0.1$. For the space S' the parameters found to best perform were $\tau_{S'} = 0.1$ and $cp_{S'} = 0.5$.

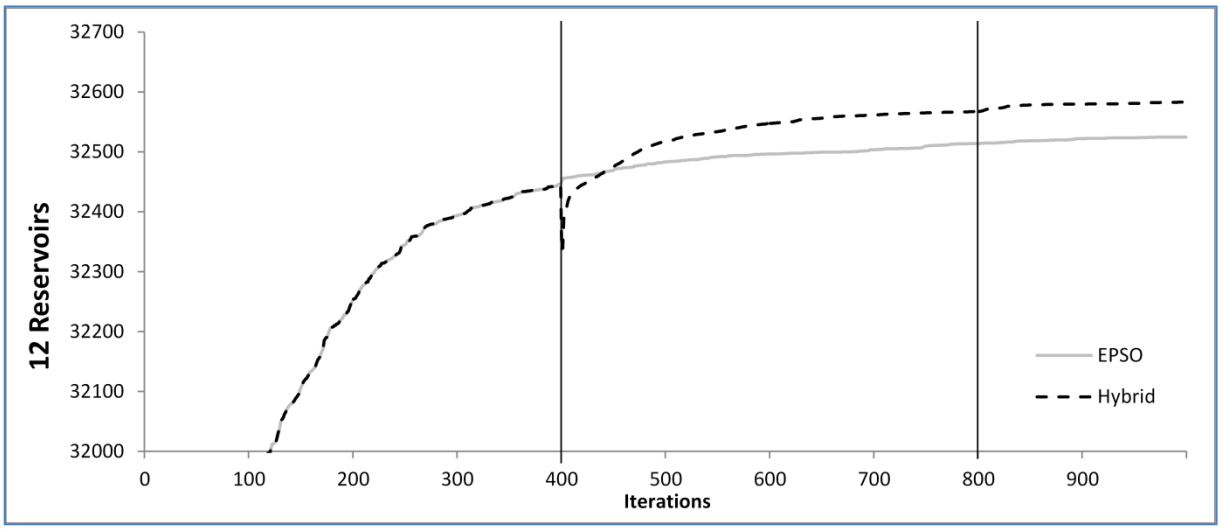


Figure R1/ 44 - Results for 12Reservoirs problem with Hybrid [dim 146 – 50 – 146] and EPSO [dim 146]. Average of 10 runs.

	ALPINE	GRIEWANK	ROSENBROCK	SPHERE	SHIFTED RASTRIGIN	SHIFTED SCHWEFEL	SHIFTED SPHERE	RES8	RES12
Tau S	0.4	0.6	0.8	0.7	0.9	0.8	0.7	0.9	0.9
CP S	0.1	0.95	0.9	0.9	0.9	0.9	0.9	0.7	0.1
Tau S'	0.4	0.7	0.6	0.8	0.8	0.1	0.8	0.8	0.1
CP S'	0.95	0.5	0.5	0.5	0.4	0.9	0.5	0.1	0.5
swarm	400	400	200	200	400 (200 for DIM experiments)	400	400	50	50
iterations S+S'+S	100+100+100	15+50+100	100+50+150	100+100+100	100+30+100	10+10+60	100+100+100	400+400+200	400+400+200
Method Train	Prop	ITL	ITL	ITL	Prop	ITL	ITL	Prop	Prop
ActFn 2nd	tan	tan	tan	tan	tan	tan	tan	tan	tan
ActFn 3rd	lin	lin	lin	tan	lin	lin	tan	lin	lin
Normalization	Global MinMax	Entrance MinMax	Entrance MinMax	Global MinMax	Global MinMax	Entrance MinMax	Global MinMax	Entrance MinMax	Global MinMax
fbias	---	---	---	---	-330	-450	-450	---	---
Computational Average Time (in seconds) (for 10 seeds)	847.8	4784	4777.7	8570	886	7160.6	7468.3	790.6	1172

Figure R1/ 45 - Summary of main specifications for the experiments presented.

19 Bibliography

Chang, S. C., Chen, C. H., Fong, I. K., & Luh, P. B. (1990). Hydroelectric generation scheduling with an effective differential dynamic programming algorithm. *IEEE Transactions on Power Systems, 5*(3), 737-743.

Clerc, M. (1998). The Alpine function. from http://clerc.maurice.free.fr/pso/Alpine/Alpine_Function.htm

Costa, L. (2008). *Application of Evolutionary Swarms and Autoencoders to Wind-Hydro coordination.* Faculty of Engineering of the University of Porto, Porto, Portugal.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design*. Boston and London: Pws Pub.

Hedar, A. (2012a). Test Functions for Unconstrained Global Optimization - Griewank Function. Retrieved May, 27, 2013, from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2537.htm

Hedar, A. (2012b). Test Functions for Unconstrained Global Optimization - Schwefel Function. Retrieved May, 27, 2013, from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2537.htm

Hedar, A. (2012c). Test Functions for Unconstrained Global Optimization - Sphere Function. Retrieved May, 27, 2013, from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2537.htm

Jolliffe, I. T. (2002). *Principal Component Analysis*. New York: Springer Series in Statistics, Springer.

Labadie, J. W. (2004). Optimal Operation of Multireservoir Systems: State-of-the-Art Review. *Journal of Water Resources Planning and Management, 130*(2), 93-111.

Lyra, C., Tavares, H., & Soares, S. (1984). Modelling and Optimization of Hydrothermal Generation Scheduling. *IEEE Transaction on Power Apparatus and Systems, 103*(8), 2126-2133.

MathWorks. (2012). Documentation Center - Sphere. Retrieved May 22, 2013, from http://www.mathworks.com/help/matlab/ref/sphere.html

Palma, V., & Hora, J. (2012). Theoretical Concepts of ITL Neural Networks *INESC Interim Report*. Porto, Portugal.

Pan, Q. K., Suganthan, P., Tasgetiren, M. F., & Liang, J. (2010). A self-adaptive global best harmony search algorithm for continuous optimization problems. *Applied Mathematics and Computation, 216*(3), 830-848.

Rao, R. V., & Savsani, V. J. (2012). Mechanical design optimization using advanced optimization techniques: Springer.

Richling, M. (2009). MR. Retrieved May 29, 2013, from http://www.mitchr.me/SS/mjrcalc/lispy/exClassicOptBanana-ART.png.html

Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function. *Computer Journal, 3*(3), 175-184.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by back-propagating errors. *Letters to Nature, 323*(9), 533-536.

Soares, S., & Carneiro, A. (1991). Optimal Operation of Reservoirs for Electric Generation. *IEEE Transactions on Power Delivery, 6*(3), 1101-1107.

Soares, S., Lyra, C., & Tavares, H. (1980). Optimal Generation Scheduling of Hydrothermal Power Systems. *IEEE Transactions on Power Apparatus and Systems, 99*(3), 1107-1118.

US DIBRPRO. (2005). Managing Water in the West - Hydroelectric Power: U. S. Department of the Interior Bureau of Reclamation Power Resources Office (US DIBRPRO).

Weisstein, E. W. (2012). MathWorld--A Wolfram Web Resource. from http://mathworld.wolfram.com/GriewankFunction.html

Wikipedia. (2012a). File: Rastrigin function.png. Retrieved April 12, 2013, from http://en.wikipedia.org/wiki/File:Rastrigin_function.png

Wikipedia. (2012b). Rosenbrock_function. from http://en.wikipedia.org/wiki/Rosenbrock_function

Zoumas, C. E., Bakirtzis, A. G., Theocharis, J. B., & Petridis, V. (2004). A Genetic Algorithm Solution Approach to the Hydrothermal Coordination Problem. *IEEE Transactions on Power Systems, 19*(2), 1356-1364.

Breakers' state estimation using autoassociative neural networks

PTDC/EEA-EEL/104278/2008

Report LASCA / R2

20 Abstract

This work addresses the problem of breakers' topology estimation in power systems. A classification method is applied, which considers the competition of two autoencoders, each one trained to learn a specific manifold concerning a breaker's status: "open" and "closed". The classification decision is made with the adoption of the state referring to the autoencoder with the lowest error. The methodology was applied for 11 breakers, under two modes: the first includes, for the training of the autoencoders, all information that is immediately adjacent to the breaker and its buses, the second excludes information concerning the flows directly connected to the respective breaker. The first technique returned performances that can be considered equivalent to the empirical approach. The second approach tested (without direct flows), represents a significant gain: without information on direct flows, the empirical approach is unfeasible. Impressively, this approach achieved 100.00% of accuracy for 4 breakers, and an accuracy of 98.16% for the worst breaker.

21 1 Introduction

This work is conducted under the research field of breakers' topology estimation in power systems. The methodology applied considers two autoencoders trained for each breaker: i) one autoencoder is trained using instances with status "open", and ii) a second autoencoder with status "closed". Both autoencoders are further applied together as a classifier, following a competitive procedure. Each instance to be classified (from validation dataset) passes through each autoencoder, being the respective error stored. The classification decision is made with the adoption of the state referring to the autoencoder with the lowest error. Previous works addressing a similar methodology are (Miranda, Krstulovic, Keko, Moreira, & Pereira,

2012), (Krstulovic, Miranda, Costa, & Pereira, 2013) and (Miranda, Krstulovic, Hora, Palma, & Príncipe, 2013).

This work explores two case studies: the first encompasses 10 breakers (Breakers 1 to 10), and the second 1 breaker (Breaker 11). The dataset concerning Breakers 1 to 10 is the same as that of study (Krstulovic et al., 2013), and concerning Breaker 11 is the same as that of study (Miranda et al., 2013). Therefore, the results here reported are directly comparable. These data were obtained with simulation of IEEE benchmark network (IEEE RTS Task Force of APM Subcommittee, 1979). The autoencoders used in this work were implemented in C++ by the authors. Details concerning the methods employed can be found in (Palma & Hora, 2012; Palma & Martins, 2012; Palma & Martins, 2013).

The main contributions of this work are: **i)** the employment of autoencoders trained with ITL concepts to address the problem of breakers' state estimation (section 5); **ii)** the proposal of a new approach able to accurately estimate the breakers' state without information on flows directly connected to the breakers (section 6).

This working paper is organized as follows. Section 2 describes the problem. Section 3 summarizes some measures adequate to assess the performance of classifiers. Section 4 provides the classification results obtained with an empirical method which work with the establishment of thresholds for the values observed in the flows where breakers are set. The classification results obtained with the competitive ITL autoencoders are summarized in Section 5. Section 6 includes the classification results using competitive ITL autoencoders whose train does not include the flows directly connected to the breakers. Section 7 provides the discussion and conclusions.

22 2 Problem Description

22.1 2.1 Case study 1

Case study 1 addresses the topology estimation on 10 switchers. The electrical scheme for these switchers is provided in Figure R2/ 1. Details on this load model can be found in (IEEE RTS Task Force of APM Subcommittee, 1979).

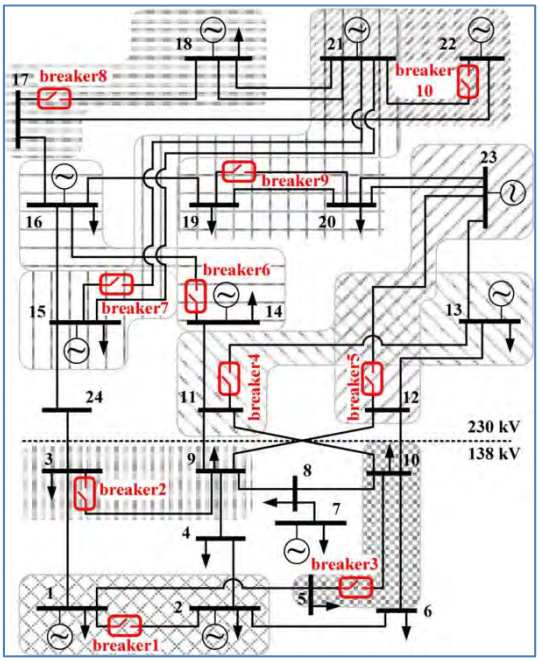


Figure R2/ 1 - Case study 1 power system scheme, IEEE RTS 24, with identification of Breakers 1 to10, from (Krstulovic et al., 2013).

For each breaker, the variables considered include the active and reactive power injected on the buses adjacent to the breaker, and active and reactive power of flows directly connected to the adjacent buses. For Breaker 1, the variables considered are: the injected active and reactive power from buses 1 and 2 (*P_inj1*, *Q_inj1*, *P_inj2* and *Q_inj2*), the active and reactive power of flows directly connected to buses 1 and 2: flow [1-3] (*P_flow1-3* and *Q_flow1-3*), flow [1-5] (*P_flow1-5* and *Q_flow1-5*), flow [1-2] (*P_flow1-2* and *Q_flow1-2*), flow [2-4] (*P_flow2-4* and *Q_flow2-4*) and flow [2-6] (*P_flow2-6* and *Q_flow2-6*). Analogous reasoning is applied for all switchers. Table 1 provides the summary of variables considered for each breaker, where the cells shaded with green indicate the flows directly connected to the respective breaker. A note must be made concerning "double flows", when two buses are connected with more than one flow (e.g. flow [18-21]). For those cases, the first flow is designated normally ("*P_flow18-21*") and the second is designated with the same label ending with the number 2 ("*P_flow18-212*").

Table R2/ 1 - Variables used for Breakers 1 to 10.

Breaker 1 switch_1-2	Breaker 2 switch_3-9	Breaker 3 switch_5-10	Breaker 4 switch_11-13	Breaker 5 switch_12-23	Breaker 6 switch_14-16	Breaker 7 switch_15-21	Breaker 8 switch_17-18	Breaker 9 switch_19-20	Breaker 10 switch_21-22

P_inj1	P_inj3	P_inj5	P_inj11	P_inj12	P_inj16	P_inj15	P_inj17	P_inj19	P_inj21
Q_inj1	Q_inj3	Q_inj5	Q_inj11	Q_inj12	Q_inj16	Q_inj15	Q_inj17	Q_inj19	Q_inj21
P_inj2	P_inj9	P_inj10	P_inj13	P_inj23	P_inj14	P_inj21	P_inj18	P_inj20	P_inj22
Q_inj2	Q_inj9	Q_inj10	Q_inj13	Q_inj23	Q_inj14	Q_inj21	Q_inj18	Q_inj20	Q_inj22
P_flow 1-3	P_flow1 -3	P_flow1- 5	P_flow1 1-14	P_flow9 -12	P_flow1 1-14	P_flow15 -16	P_flow1 7-18	P_flow1 6-19	P_flow18 -21
Q_flow 1-3	Q_flow 1-3	Q_flow1 -5	Q_flow1 1-14	Q_flow9 -12	Q_flow1 1-14	Q_flow1 5-16	Q_flow1 7-18	Q_flow1 6-19	Q_flow1 8-21
P_flow 2-4	P_flow3 -24	P_flow5- 10	P_flow9 -11	P_flow1 0-12	P_flow1 4-16	P_flow15 -24	P_flow1 6-17	P_flow1 9-20	P_flow18 -212
Q_flow 2-4	Q_flow 3-24	Q_flow5 -10	Q_flow9 -11	Q_flow1 0-12	Q_flow1 4-16	Q_flow1 5-24	Q_flow1 6-17	Q_flow1 9-20	Q_flow1 8-212
P_flow 1-2	P_flow3 -9	P_flow6- 10	P_flow1 0-11	P_flow1 2-13	P_flow1 6-19	P_flow15 -21	P_flow1 7-22	P_flow1 9-202	P_flow15 -21
Q_flow 1-2	Q_flow 3-9	Q_flow6 -10	Q_flow1 0-11	Q_flow1 2-13	Q_flow1 6-19	Q_flow1 5-21	Q_flow1 7-22	Q_flow1 9-202	Q_flow1 5-21
P_flow 2-6	P_flow4 -9	P_flow8- 10	P_flow1 1-13	P_flow1 2-23	P_flow1 6-17	P_flow15 -212	P_flow1 8-21	P_flow2 0-23	P_flow15 -212
Q_flow 2-6	Q_flow 4-9	Q_flow8 -10	Q_flow1 1-13	Q_flow1 2-23	Q_flow1 6-17	Q_flow1 5-212	Q_flow1 8-21	Q_flow2 0-23	Q_flow1 5-212
P_flow 1-5	P_flow8 -9	P_flow1 0-11	P_flow1 3-23	P_flow1 3-23	P_flow1 5-16	P_flow18 -21			P_flow21 -22
Q_flow 1-5	Q_flow 8-9	Q_flow1 0-11	Q_flow1 3-23	Q_flow1 3-23	Q_flow1 5-16	Q_flow1 8-21			Q_flow2 1-22

	P_flow9-11	P_flow10-12	P_flow12-13	P_flow20-23		P_flow21-22	P_flow17-22
	Q_flow9-11	Q_flow10-12	Q_flow12-13	Q_flow20-23		Q_flow21-22	Q_flow17-22
	P_flow9-12						
	Q_flow9-12						

Information for empirical approach

The empirical approach uses only the information concerning the respective flows of the breaker under study (the cells shaded with green in Table R2/ 1).

Information for autoencoders with all variables

The train of autoencoders includes all variables detailed in Table R2/ 1 for each breaker. Moreover, the number of neurons composing each layer is summarized in Table R2/ 2.

Table R2/ 2 - Number of neurons composing each layer (autoencoders with all variables).

Layer	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
Input	14	18	16	16	16	14	16	12	12	16
Hidden	10	14	12	12	12	10	12	8	8	12
Output	14	18	16	16	16	14	16	12	12	16

Information for autoencoders without direct flows

The autoencoders which address a breaker without direct flows include all variables not shaded in Table R2/ 1. The number of neurons considered for each layer is detailed in Table R2/ 3.

Table R2/ 3 - Number of neurons composing each layer (autoencoders without direct flows).

Layer	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

Input	12	16	14	14	14	12	14	10	10	14
Hidden	8	14	10	10	10	8	10	6	6	10
Output	12	16	14	14	14	12	14	10	10	14

Data

The number of instances composing each dataset is detailed in Table R2/ 1. The classification results presented always refer to the validation dataset, accounting in 10000 instances for each breaker, concerning instances of status *open* and *closed* (*Validation_O* and *Validation_C*). The train and test datasets are used to train each autoencoder: the train instances allow the autoencoder to learn the respective manifold, and the test dataset ensures the rejection of overfitted models. More details concerning theoretical aspects on validation, train and test datasets can be found in (Palma & Martins, 2013).

Table R2/ 4 - Number of examples composing each dataset.

Dataset		B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
Open	Train_O	4058	4060	4009	4076	3965	3887	3992	4093	3998	4004
	Test_O	1006	980	998	978	1003	983	991	1001	953	987
	Validation_O	4956	4987	4992	5056	4961	4948	5016	5010	5061	4943
Closed	Train_C	3942	3940	3991	3924	4035	4113	4008	3907	4002	3996
	Test_C	994	1020	1002	1022	997	1017	1009	999	1047	1013
	Validation_C	5044	5013	5008	4944	5039	5052	4984	4990	4939	5057

22.2 2.2 Case study 2

Case study 2 addresses a distinct IEEE RTS 24 system from case study 1. These two systems differ from the position of breaker 3: in case study 2 the breaker 3 is positioned between buses 6 and 10 (whilst in case study 1, breaker 3 was positioned between buses 5 and 10). The system for case study 2 is represented in Figure R2/ 2.

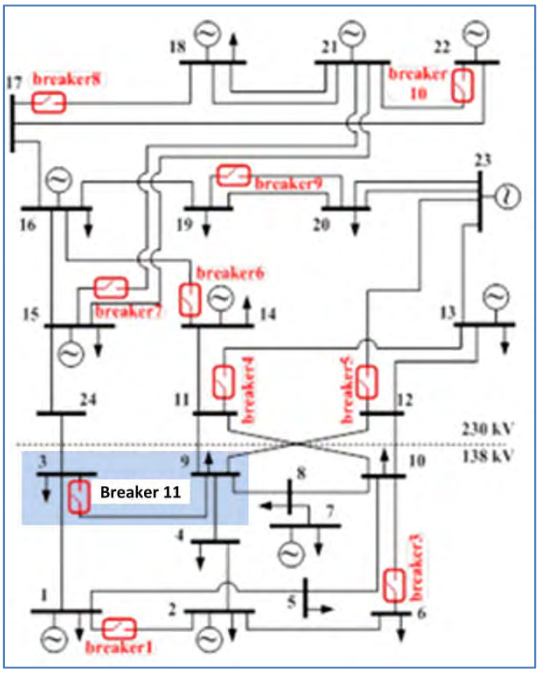


Figure R2/ 2 - Case study 2 power system scheme, IEEE RTS 24, with identification of Breaker 11.

In this case study, Breaker 11 is the only one to be explored. This decision was made as Breaker 11 revealed to be a remarkably difficult breaker concerning state estimation.

Information for empirical approach

The information to use under the empirical approach concerns the active and reactive power on flow [3-9] (i.e. *P_flow3-9* and *Q_flow3-9*).

Information for autoencoders with all variables

The information used under the general autoencoders approach includes P_inj3, Q_inj3, P_inj9, Q_inj9, P_flow1-3, Q_flow1-3, P_flow3-24, Q_flow3-24, P_flow3-9, Q_flow3-9, P_flow4-9, Q_flow4-9, P_flow8-9, Q_flow8-9, P_flow9-11, Q_flow9-11, P_flow9-12 and Q_flow9-12.

Therefore, the number of neurons integrating the input and output layers is 18, and for the hidden layer 13 neurons were considered.

Information for autoencoders without direct flows

The variables included within the implementation of autoencoders without information concerning the respective flows are: *P_inj3*, *Q_inj3*, *P_inj9*, *Q_inj9*, *P_flow1-3*, *Q_flow1-3*, *P_flow3-24*, *Q_flow3-24, P_flow3-9*, *Q_flow3-9, P_flow4-9, Q_flow4-9, P_flow8-9, Q_flow8-9, P_flow9-11, Q_flow9-11, P_flow9-12 and Q_flow9-12*.

Concerning the number of neurons, the input and output layers include 16 neurons each, and the hidden layer 11 neurons.

23 3 Measuring performance in classification

The assessment of performance of classifiers is usually made by means of Receiver operating characteristics (ROC) analysis. An elucidative guide to ROC analysis is provided in (Fawcett, 2006). The confusion matrix includes information concerning true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN), as exemplified in Eq. R2/(1).

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \quad \text{Eq. R2/(1)}$$

Other important measures, which are calculable from confusion matrix, are Area Under ROC Curve (AUC), precision, recall, accuracy and specificity. The detail on these measures is provided in expressions Eq. R2/(2) to Eq. R2/(8).

$$P = TP + FN \quad \text{Eq. R2/(2)}$$

$$N = FN + TN \quad \text{Eq. R2/(3)}$$

$$Precision = \frac{TP}{TP + FP} \quad \text{Eq. R2/(4)}$$

$$TP_{rate} = recall = hit_{rate} = \frac{TP}{P} \quad \text{Eq. R2/(5)}$$

$$FP_{rate} = \frac{FP}{N} \quad \text{Eq. R2/(6)}$$

$$accuracy = \frac{TP + TN}{P + N} \quad \text{Eq. R2/(7)}$$

$$specificity = \frac{TN}{FP + TN} = 1 - FP_{rate} \quad \text{Eq. R2/(8)}$$

24 4 Topology estimation using empirical approach

The empirical approach described in this section refers to the procedure used in the field to estimate whether a breaker is open or closed. This approach works with the establishment of thresholds for the

active and reactive power values observed in flows where breakers are allocated. This approach starts by selecting minimum and maximum limits observed for the train dataset concerning reactive and active power flows, these limits are selected considering only the examples where the breaker's status was "open". Having these limits selected, each instance from the validation dataset is classified as "open" if its reactive power flow value is within the respective limits and if its active power flow value is also within the respective limits. Otherwise, the instance is classified as "closed".

There are some variants of this main idea: **i)** consider the limits obtained with the inclusion of some delta; **ii)** adopting of the value zero as a reference, consider the maximum absolute value observed to define the variance, this way originating a symmetric interval (noting that the process firstly describe originate asymmetric intervals). At this point, the process followed within the empirical approach is presented. Next sections include the results obtained considering symmetric limits, asymmetric limits, both with and without deltas.

24.1 4.1 Case Study 1

The results obtained with the empirical approach concerning the Breakers 1 to 10 are summarized in Table R2/ 6. Breakers 1, 2, 4, 5, 6, 7 and 10 achieved an accuracy of 100.00%. The worst case was Breaker 9, for which the best results obtained was of 11 fails in 10000 instances, corresponding to an accuracy of 99.89%. For Breaker 3, the best result obtained was of 2 fails (accuracy of 99.98%), and for Breaker 8 the best result was of 1 fail in 10000 instances (accuracy of 99.99%).

24.2 4.2 Case Study 2

Concerning Breaker 11, the results obtained are summarized in Table R2/ 5.

Table R2/ 5 - Results of empirical approach for Breaker 11.

Asymmetric empirical approach					Symmetric empirical approach				
Delta	TN	FP	FN	TP	Delta	TN	FP	FN	TP
-0.01	665	4402	0	4933	-0.01	1342	3725	0	4933
-0.005	4581	486	2	4931	-0.005	4785	282	2	4931

-0.0005	5052	15	6	4927	-0.0005	5059	8	6	4927
0	5060	7	6	4927	0	5063	4	6	4927
0.0005	5062	5	7	4926	0.0005	5067	0	7	4926
0.005	5067	0	11	4922	0.005	5067	0	12	4921
0.01	5067	0	29	4904	0.01	5067	0	31	4902

Using the empirical approach, the best result obtained for Breaker 11 was of 7 fails in 10000 instances, corresponding to an accuracy of 99.93%.

25 5 Competitive autoencoders with all variables

This section includes the results obtained with the application of competitive autoencoders to estimate breakers' status in power systems. Different specifications of autoencoders' parameters were tested. All experiments considered the initialization of synaptic weights of 1st half (W1) with PCA, the initialization of synaptic weights of 2nd half (W2) with the transposition of weights from 1st half, the initialization of bias with zeros, normalization using the min max by entrance. The normalization of synaptic weights using the Oja's rule was also tested (Oja, 1982; Palma & Martins, 2013). A detailed description on the meaning of these specifications can be found in (Palma & Hora, 2012; Palma & Martins, 2013). Note that the initialization adopted ensures the randomness independence of these autoencoders (meaning that a single run ensures the results presented).

25.1 5.1 Case Study 1

The results obtained with the application of competitive autoencoders to estimate the state of Breakers 1 to 10 is summarized in Table R2/ 7.

Breakers 1, 2, 4, 5, 6, 7 and 10 achieved 100% of accuracy for at least one experiment. Breaker 9 revealed to be the most difficult one, for which the best result obtained was out of 12 failures in 10000 instances using PROP, corresponding to an accuracy of 99.88%. Using ITL autoencoders, the best accuracy found for Breaker 9 was 99.82% (18 fails in 10000 instances).

Table R2/ 6 - Results of empirical approach for Breakers 1 to 10.

		Assymmetric empirical approach							Symmetric empirical approach						
Breaker 1	Delta	-0	-0.01	-5.00E-04	0	5.00E-04	0.01	0.01	-0	-0.01	-5.00E-04	0	5.00E-04	0.01	0.01
	TN	1175	4607	4950	4956	4957	4957	4957	1733	4758	4956	4957	4957	4957	4957
	FP	3782	350	7	1	0	0	0	3224	199	1	0	0	0	0
	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	5044	5044	5044	5044	5044	5044	5044	5044	5044	5044	5044	5044	5044	5044
Breaker 2	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	662	4512	4980	4982	4983	4987	4987	705	4536	4980	4983	4983	4987	4987
	FP	4325	475	7	5	4	0	0	4282	451	7	4	4	0	0
	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	5013	5013	5013	5013	5013	5013	5013	5013	5013	5013	5013	5013	5013	5013

Breaker 3	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	853	4571	4985	4988	4989	4992	4992	1103	4653	4986	4988	4989	4992	4992
	FP	4139	421	7	4	3	0	0	3889	339	6	4	3	0	0
	FN	0	2	2	2	2	2	3	0	2	2	2	2	2	3
	TP	5008	5006	5006	5006	5006	5006	5005	5008	5006	5006	5006	5006	5006	5005
Breaker 4	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	995	4673	5048	5053	5055	5056	5056	1557	4822	5053	5055	5056	5056	5056
	FP	4061	383	8	3	1	0	0	3499	234	3	1	0	0	0
	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	4944	4944	4944	4944	4944	4944	4944	4944	4944	4944	4944	4944	4944	4944
Breaker 5	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	842	4560	4952	4956	4957	4961	4961	1444	4715	4959	4960	4960	4961	4961
	FP	4119	401	9	5	4	0	0	3517	246	2	1	1	0	0

	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	5039	5039	5039	5039	5039	5039	5039	5039	5039	5039	5039	5039	5039	5039
Breaker 6	Delta	-0	-0.01	-5.00E-04	0	5.00E-04	0.01	0.01	-0	-0.01	-5.00E-04	0	5.00E-04	0.01	0.01
	TN	812	4551	4936	4941	4944	4948	4948	1129	4600	4940	4943	4945	4948	4948
	FP	4136	397	12	7	4	0	0	3819	348	8	5	3	0	0
	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	5052	5052	5052	5052	5052	5052	5052	5052	5052	5052	5052	5052	5052	5052
Breaker 7	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	266	4356	5004	5010	5011	5016	5016	522	4536	5009	5013	5014	5016	5016
	FP	4750	660	12	6	5	0	0	4494	480	7	3	2	0	0
	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	4984	4984	4984	4984	4984	4984	4984	4984	4984	4984	4984	4984	4984	4984
Breaker 8	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	155	4760	5007	500	5009	5010	501	185	4802	5009	500	5009	5010	501

		4			8			0	9			9			0
	FP	3456	250	3	2	1	0	0	3151	208	1	1	1	0	0
	FN	0	0	0	0	0	1	1	0	0	0	0	0	1	1
	TP	4990	4990	4990	4990	4990	4989	4989	4990	4990	4990	4990	4990	4989	4989
Breaker 9	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	875	4637	5054	5054	5058	5061	5061	1830	4884	5060	5060	5060	5061	5061
	FP	4186	424	7	7	3	0	0	3231	177	1	1	1	0	0
	FN	0	1	7	8	8	14	21	1	1	10	10	10	15	22
	TP	4939	4938	4932	4931	4931	4925	4918	4938	4938	4929	4929	4929	4924	4917
Breaker 10	Delta	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01	-0	-0.01	-5.00E-04	0	0.0005	0.01	0.01
	TN	945	4595	4940	4940	4942	4943	4943	1260	4677	4940	4940	4943	4943	4943
	FP	3998	348	3	3	1	0	0	3683	266	3	3	0	0	0
	FN	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TP	5057	5057	5057	5057	5057	5057	5057	5057	5057	5057	5057	5057	5057	5057

Table R2/ 7 - Number of fails obtained with competitive autoencoders using all variables for Breakers 1 to 10.

Activation functions	Oja	W1	W2	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
Tan - lin	No	Prop		0	3	303	0	0	0	0	311	12	1
	Yes			2	197	455	2	0	0	46	586	439	0
Tan - Tan	Yes	CS	Prop	117	1	34	1	0	0	39	1146	755	1
			Corr	45	0	5	0	0	0	2	430	18	39
	No		Prop	2129	3	522	0	0	0	0	675	28	1
			Corr	0	0	870	0	0	0	0	2	26	0
Tan - Lin	Yes		Prop	611	110	112	0	0	0	169	307	485	0
			Corr	1011	233	111	1	0	0	0	6	472	0
	No		Prop	2535	249	487	0	0	0	26	1609	31	1
			Corr	3935	0	411	0	0	0	5124	2	31	5
Lin -Lin	Yes		Prop	6	1958	141	0	0	8	9	1623	387	1
			Corr	7	2428	97	0	0	18	24	670	758	1

	No		Prop	2	2015	1723	0	0	22	54	3308	650	1
			Corr	73	1710	149	32	0	43	25	3320	930	1

25.2 5.2 Case Study 2

Concerning Breaker 11, the results obtained are detailed in Table R2/ 8. The experiments here reported differ from the ones reported in (Miranda et al., 2013) concerning the number of instances used to perform the train of the autoencoders: in this study the we used approximately 4000 instances to train each autoencoder and 1000 to assess overfitting, while in (Miranda et al., 2013) 1000 instances were used to train each autoencoder and 500 to assess overfitting. Nevertheless, the 10000 instances composing the validation dataset are the same.

Table R2/ 8 - Number of fails obtained with competitive autoencoders using all variables for Breaker 11.

Activation functions	Oja	W1	W2	B11
Tan - lin	No	Prop		288
	Yes			1550
Tan - Tan	Yes	CS	Prop	2862
			Corr	2593
	No		Prop	1093
			Corr	1347
Tan - Lin	Yes		Prop	1716
			Corr	1904
	No		Prop	647

			Corr	442
Lin -Lin	Yes		Prop	242
			Corr	17
	No		Prop	572
			Corr	**11**

The best result obtained corresponds to an accuracy of 99.89% (11 fails out of 10000 instances). Nevertheless, the majority of experiments returned high values of fails, indicating that this breaker is a difficult case to estimate.

26 6 Competitive autoencoders without direct flows

This section includes the results obtained by considering the competition of two autoencoders, each one trained for a different status (i.e. one autoencoder is trained with instances referring to the status “open”, and the other is trained to reproduce the “closed” status).

All experiments considered initialization of synaptic weights of 1st half (W1) with PCA, the initialization of synaptic weights of 2nd half (W2) with the transposition of the synaptic weights of the 1st half. The initialization of bias was always specified with zeros. This initialization process is completely independent from randomness, from where one single run produces the results reported. Theoretical details concerning these specifications are detailed in (Palma & Hora, 2012).

Concerning the tables further presented, the label “tan-lin” indicates that the activation functions used in 2nd and 3rd layers were hyperbolic tangent and piecewise-linear, respectively; the label “tan-tan” indicates that the hyperbolic tangent activation function was used on both 2nd and 3rd layers. The employment of Oja’s rule (Oja, 1982; Palma & Martins, 2013) is shown with “yes”, and its absence with “no”, on the respective line. The methods applied to train the synaptic weights of 1st (W1) and 2nd halves of the autoencoders are: maximization of entropy between two consecutive layers with estimators of Cauchy Schwarz (CS), the minimization of correntropy between two consecutive layers (Corr) or the minimization of mean squared error with backpropagation (PROP). The estimation of sigma was as proposed by Silverman (Silverman, 1986). More details on the theory of these specifications are available at (Palma & Hora, 2012; Palma & Martins, 2013).

26.1 6.1 Case study 1

The number of fails obtained for each breaker is detailed in Table R2/ 9.

Table R2/ 9 - Number of fails obtained with competitive autoencoders without direct flows for Breakers 1 to 10.

Breaker	1	2	3	4	5	6	7	8	9	10
Activation functions	tan-lin / tan-tan	tan-lin	tan-lin	tan-lin	tan-lin	tan-lin	tan-lin	tan-lin	tan-lin	tan-tan
Oja	no	yes	yes	no	yes	no/yes	no	no	no	no
W1	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
W2	PROP	PROP	PROP	PROP / Corr	PROP / Corr	PROP / Corr	Corr	PROP	Corr	PROP
Data normalization	Global minmax	Entrance minmax	Entrance minmax	Entrance minmax	Entrance minmax	Entrance minmax	Entrance minmax	Entrance minmax	Entrance minmax	Entrance minmax
Best no. fails	0	143	18	0	0	0	14	42	184	2

Breakers 1, 4, 5 and 6 returned 100.00% of accuracy. Once again, the state of Breaker 9 was the most difficult to estimate, with an accuracy of 98.16%.

26.2 6.2 Case study 2

Concerning Breaker 11, the number of fails is presented in Table R2/ 10, alongside with some details concerning the specifications used. For this case, the best accuracy found was of 98.41%, with 159 fails out of 10000 instances.

Table R2/ 10 - Number of fails obtained with competitive autoencoders without direct flows for Breaker 11.

Breaker	11
Activation functions	tan-lin

Oja	No
W1	CS
W2	PROP
Data normalization	Global minmax
Best no. fails	159

27 7 Discussion and Conclusions

The employment of the technique *competitive autoencoders with all variables* returned good results: 7 cases returned a performance of 100.00%, and the worst breaker returned a performance of 99.89%. However, a significant gain cannot be concluded from these results when compared with the accuracy achieved from the empirical approach.

The second approach tested, the use of *competitive autoencoders without direct flows*, represents a significant gain, as it allows the breakers' status estimation using only indirect information concerning the respective breaker. Without information on direct flows, the empirical approach is unfeasible. Impressively, this approach achieved 100.00% of accuracy for 4 breakers, having the worst breaker returned an accuracy of 98.16%.

28 Bibliography

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*, 861-874.

IEEE RTS Task Force of APM Subcommittee. (1979). IEEE Reliability Test System. *IEEE Transactions on PAS, 98*(6), 2047-2054.

Krstulovic, J., Miranda, V., Costa, A. J. A. S., & Pereira, J. (2013). Towards an auto-associative topology state estimator. *IEEE Transactions on Power Systems, 28*(3), 3311 - 3318

Miranda, V., Krstulovic, J., Hora, J., Palma, V., & Príncipe, J. C. (2013). *Breaker status uncovered by autoencoders under unsupervised maximum mutual information training.* Paper presented at the ISAP 2013, Japan.

Miranda, V., Krstulovic, J., Keko, H., Moreira, C., & Pereira, J. (2012). Reconstructing Missing Data in State Estimation With Autoencoders. *IEEE Transactions on Power Systems, 27*(2), 604-611.

Oja, E. (1982). A Simplified Neuron Model as a Principal Component Analyser. *Journal of Mathematical Biology, 15*(3), 267-273.

Palma, V., & Hora, J. (2012). Theoretical Concepts of ITL Neural Networks *INESC Interim Report*. Porto, Portugal.

Palma, V., & Martins, J. H. (2012). Theoretical Concepts of BackPropagation Neural Networks *INESC Interim Report*. Porto, Portugal: INESC TEC.

Palma, V., & Martins, J. H. (2013). Training Neural Networks - Theory of Practical Issues *INESC Interim Report*. Porto, Portugal: INESC TEC.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. London, UK: Chapman & Hall/CRC.

On the equivalence of maximizing entropy and mutual information

PTDC/EEA-EEL/104278/2008

Report LASCA / R3

29 Abstract

This study is conducted under the context of unsupervised training of neural networks with two layers, using the concepts of information theory to perform the training. The two criteria here addressed are: i) maximizing the entropy of the outputs (MaxEnt) and ii) maximization the mutual information between the inputs and outputs (MaxMI). The research question pursued is *"are these two approaches equivalent?"*. With base on the existing literature, it is possible to conclude that the two approaches are theoretically equivalent provided the system is noiseless.

30 1 Introduction

Let us consider a neural network composed of two layers. The first layer, the input, is represented by X. The second layer, the output layer, is represented by Y. The number of neurons composing each layer is arbitrary, and we assume the general case where the two layers are composed with a distinct number of neurons. Figure 1 includes a schematic of the neural network considered.

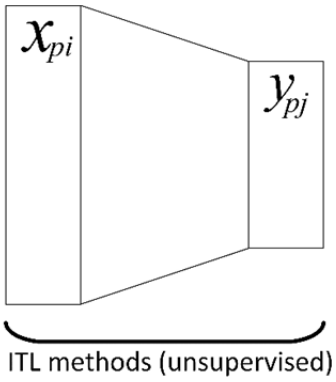


Figure R3/ 1 – Schematic representation of the first half of an autoencoder to be trained with ITL criteria.

The neural network is to be trained following an unsupervised approach using one information criterion. In this study we consider two information criteria: the first is the maximization of the mutual information between the first and the second layer $I(X,Y)$. The second information criterion is the maximization of the entropy of the outputs $H(X)$.

The study intends to study whether the two criteria are equivalent to train the neural network. The case of applying Shannon estimators is explored in the section 2. The quadratic information of Renyi is explored in section 3. Theoretical comparisons on particular aspects concerning the definitions of Mutual Information by Shannon and by Renyi are presented at section 4, and concerning the definitions of Entropy at section 5. The discussion and conclusions are presented at section 6.

31 2 Shannon estimators

The mutual information between the inputs and outputs is related with the entropy of the outputs considering the following expressions (Cover & Thomas, 1991):

$$I_S(X,Y) = H_S(X) + H_S(Y) - H_S(X,Y)$$

$$I_S(X,Y) = H_S(Y) - H_S(Y|X)$$

To compare the unsupervised training criterion of maximizing $H(Y)$ with the criterion of maximizing $I(X,Y)$, the derivatives of both concerning the weights are to be studied. The study conducted by (Bell & Sejnowski, 1995) addresses this question concerning the maximization of the **"differential"** entropy, which is the entropy described by Shannon, and proves that maximizing Shannon entropy is equivalent of maximizing the mutual information between the inputs and outputs.

$$\frac{\partial I_S(X,Y)}{\partial w} = \frac{\partial H_S(Y)}{\partial w} - \frac{\partial H_S(Y|X)}{\partial w}$$

$$\frac{\partial I_S(X,Y)}{\partial w} = \frac{\partial H_S(Y)}{\partial w}$$

Bell and Sejnowski state that, whatever the level of additive noise in the neural network, both approaches (MaxEnt and MaxMI) are equivalent. This equivalence is explained by the fact that the conditional entropy is independent of the weights (Bell & Sejnowski, 1995).

32 3 Renyi's Information estimators

When applying the estimators of Renyi's with $\alpha = 2$, the relationship between mutual information and entropy is as follows:

$$I_2(X,Y) = H_2(X) + H_2(Y) - H_2(X,Y)$$

$$I_2(X,Y) = H_2(Y) - H_2(Y|X)$$

Now let us consider the derivatives of the mutual information of quadratic Renyi:

$$\frac{\partial I_2(X,Y)}{\partial w} = \frac{\partial H_2(Y)}{\partial w} - \frac{\partial H_2(Y|X)}{\partial w}$$

The two criteria (MaxEnt and MaxMI) would be equivalent if the derivative $\frac{\partial H_2(Y|X)}{\partial w}$ is null.

32.1 3.1 Renyi's Equivalence?

As viewed in section 3, the derivative of Renyi's Mutual information is given by $\frac{\partial I_2(X,Y)}{\partial w} = \frac{\partial H_2(Y)}{\partial w} - \frac{\partial H_2(Y|X)}{\partial w}$. The aim is state if the equivalence of MaxEnt and MaxMI, for Renyi's concepts of Information Theory, maintain or not. According to (Haykin S. , 1999, p. 515), $H_S(Y|X)$ conveys information about the processing noise, rather than about the system input X. Haykin (Haykin, 1999, pp. 521-525) presents three examples of neural networks: i) a NN with a single noisy neuron at output, ii) a NN with a single output corrupted by additive input noisy neurons and iii) a noiseless NN. For the noisy networks: $\frac{\partial H_2(Y|X)}{\partial w} \neq 0$. So, the equivalence is not verified. For noiseless networks, as explained in the third example, $\frac{\partial I_S(X,Y)}{\partial w} = \frac{\partial H_S(Y)}{\partial w}$, meaning that $\frac{\partial H_S(Y|X)}{\partial w} = 0$. This reasoning is also applicable for entropy as defined by Renyi, in particular for the quadratic $H_2(Y|X)$.

33 4 Comparing Shannon and Renyi's definitions of Mutual Information

The study (Hild II, Erdogmus, Torkkola, & Principe, 2006) includes the following relationship between Shannon and Renyi's quadratic mutual information:

$$I_S(X,Y) \cong H_2(Y) - H_2(Y|X) = I_2(X,Y)$$

This expression means that the mutual information as described by Shannon is approximately equal to the mutual information as described by Renyi with $\alpha = 2$. Moreover, according to (Hild II et al., 2006, p. 1387), there is no evidence that maximizing the mutual information using the quadratic Renyi's definition would

be equivalent to maximize the mutual information using the Shannon's definition. As one identity assumes distinct quantities using L1 and L2, the same reasoning is applicable to information quantities (Shannon and Renyi's quadratic mutual information).

34 5 Comparing Shannon and Renyi's estimators for Entropy

In the thesis (Erdogmus, 2002, p. 122), it is shown that for a particular pattern y_k , the derivative of the Shannon entropy estimator is equivalent to the derivative of the quadratic entropy of Renyi estimator:

$$\frac{\partial \hat{H}_{S,k}(Y)}{\partial w} = \frac{\partial \hat{H}_{2,k}(Y)}{\partial w}$$

However, this conclusion cannot be extended to a training set composed of two or more patterns. The following two equations show that the derivatives of Shannon entropy and Renyi's entropy estimators are different when considering two or more patterns. These equations can be found in the study (Erdogmus, Hild, Principe, Lazaro, & Santamaria, 2004, p. 1490).

$$\frac{\partial \hat{H}_S(Y)}{\partial w} = -\frac{1}{L}\sum_{j=1}^{L} \frac{\sum_{i=1}^{L} \kappa'_\sigma(y_j - y_i)\left(\frac{\partial y_j}{\partial w} - \frac{\partial y_i}{\partial w}\right)}{\sum_{i=1}^{L} \kappa_\sigma(y_j - y_i)}$$

$$\frac{\partial \hat{H}_2(Y)}{\partial w} = -\frac{\sum_{j=1}^{L}\sum_{i=1}^{L} \kappa'_\sigma(y_j - y_i)\left(\frac{\partial y_j}{\partial w} - \frac{\partial y_i}{\partial w}\right)}{\sum_{j=1}^{L}\sum_{i=1}^{L} \kappa_\sigma(y_j - y_i)}$$

35 6 Discussion and Conclusions

When using estimators of entropy, it is equivalent maximizing the mutual information between the inputs and outputs and maximizing the entropy of the output, when performing unsupervised training of a noiseless neural network.

This study also included some insights concerning the comparison between Shannon and Renyi's definitions of Mutual Information and the comparison between Shannon and Renyi's estimators for Entropy.

36 Bibliography

Bell, A., & Sejnowski, T. (1995). An Information-maximization approach to blind separation and blind deconvolution. *Neural Computation, 7*, 1129-1159.

Cover, T., & Thomas, J. (1991). Entropy, Relative Entropy and Mutual Information *Elements of Information Theory* (pp. 20): John Wiley & Sons.

Erdogmus, D. (2002). Information Theoretic Learning: Renyi's Entropy and Its Applications to Adaptive System Training.

Erdogmus, D., Hild, K. E., Principe, J. C., Lazaro, M., & Santamaria, I. (2004). Adaptive blind deconvolution of linear channels using Renyi's entropy with Parzen window estimation. *IEEE Transactions on Signal Processing, 52*(6), 1489-1498.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation* (2nd ed.). Ontario, Canada: Pearson Education.

Hild II, K., Erdogmus, D., Torkkola, K., & Principe, J. C. (2006). Feature Extraction Using Information-Theoretic Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*, 1385-1392.

Training Neural Networks - Theory of Practical Issues

PTDC/EEA-EEL/104278/2008

Report LASCA / R4

37 Abstract

Through the work conducted concerning the implementation of neural networks, some theoretical issues emerged. This work arises from the need to explore these issues, covering the main topics grounding the learning process and data processing. A review is provided on adequate methods of data normalization, synaptic weights normalization and initialization, window width estimation, activation functions selection, saturation and overfitting prevention and calibration of other parameters.

38 1 Introduction

The implementation of neural networks is a complex task, composed of many sub-tasks whose accuracy needs to be ensured. These sub-tasks include the data normalization procedure adopted, the choice on the number of hidden neurons, the selection of appropriate activation functions, the estimation of the window width, among many others topics. The need to accurately treat each one of these sub-tasks motivated the review on the adequate methods proposed in literature. This paper summarizes the review conducted and includes the specifications adopted in the neural networks implemented for each topic.

The report is organized as follows. Section 2 provides general contextualization of neural networks, and section 3 introduces the autoassociative neural networks. Section 4 discusses the batch and incremental learning approaches. Section 5 includes the description of the Gradient Method. Section 6 includes the parallel between supervised and unsupervised training. Section 7 explores aspects concerning the data sets used to train, test and validate the neural networks, alongside with the appropriate statistical tests to assess the populations' similarity over the same data sets. Section 8 focuses on the adequate number of instances to train a specified neural network. Section 9 reviews the methods to perform the normalization of data employed. The selection of the number of hidden neurons is explored in Section 10. Section 11 explains how to use the principal component analysis procedure to initialize the synaptic weights. In

Section 12 details the normalization of synaptic weights during the entire training process. Section 13 includes a detailed description on widely used activation functions and their corresponding derivatives. Section 14 includes theoretical aspects concerning the stable learning rate. The adaptive learning rate process is detailed in Section 15. Section 16 threats the estimation of the window width, and Section 17 the adaptive window width process. Section 18 details how to prevent the saturation of synaptic weights during the training. Section 19 details the prevention of overfitting.

39 2 Artificial Neural Network

An Artificial Neural Network, commonly referred to as Neural Network (NN), is a machine designed to model the way human brain performs / learns a particular task or function of interest (Haykin, 1999). A NN corresponds to a connectionist paradigm of information processing, including a massive parallel process of numerical computations (Miranda, 2007), through a process of learning. The basic processing element of a NN is the neuron (J. Principe, Euliano, & Lefebvre, 2000). Neurons are composed of several inputs, one output and an activation function which executes the internal processing, transforming the inputs into the output. Usually, neurons are organized in layers with unidirectional links always in a forward direction, from the input to the output of the NN (feedforward networks). A "full connected network" means that each neuron from one layer is connected to every neuron into the next layer. The connections between neurons are designated as synapsis.

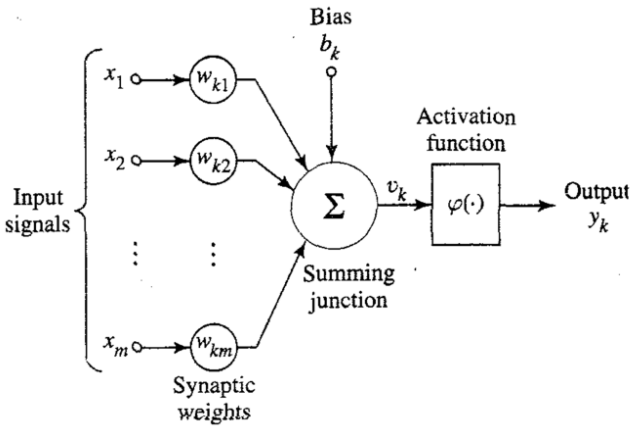


Figure R4/ 1- Nonlinear Model of a Neuron, from (Haykin S. , Neural Networks - A Comprehensive Foundation, 1999).

Each synapsis is associated with a synaptic weight w_{ij}, such that a signal emitted by a neuron is multiplied by the weight of the connection before entering a next neuron (Miranda, 2007). The bias b_k is an external parameter that allows to shift the entire curve obtained from the activation function to the left (when bias

is positive) or right (negative bias), which may be critical for successful learning (Haykin, 1999). This process is schematized in Figure R4/ 1. The equations behind the figure are:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \qquad \text{eq. R4/(1)}$$

$$v_k = u_k + b_k \qquad \text{eq. R4/(2)}$$

$$y_k = \varphi(v_k) \qquad \text{eq. R4/(3)}$$

Equation eq. R4/(1) is the weighted sum of inputs. Equation eq. R4/(2) is summing the bias b_k to the u_k. Finally, the output y_k emitted by a neuron is the result of v_k through activation function $\varphi(\cdot)$. When the input layer is intended to match the output layer, the NN is called an **autoencoder**; when the desired output differs from the input the NN is called **heteroencoder**.

40 3 Autoencoder

Auto-associative neural networks or autoencoders are feedforward networks that are trained to mirror/reproduce the input space $\boldsymbol{S}$ in the output (Miranda, Krstulovic, Keko, Moreira, & Pereira, 2012). Autoencoders define a reverse mapping consisting of a function f and its inverse f^{-1} (Figure R4/ 2) allowing to map a space of dimension m into a space of dimension n (with $n < m$) and to reconstruct the original variables (Costa, 2008).

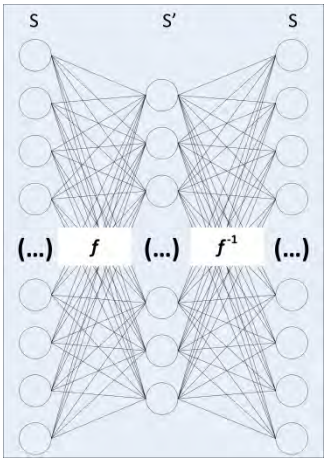


Figure R4/ 2 - Schematic representation of an autoencoder of three layers.

In general, for any NN, it is usual to call “bottleneck” to the smallest (with lowest number of neurons) hidden layer, in the case the NN is compressing information. The information contained in the entire input space is projected onto the smaller layer (with lower dimension than the input layer).

41 4 Batch versus Incremental

Through learning, the weights of the NN are adapted to learn from its environment. The environment consists of a number of given examples - the training set. The type of learning is determined by manner in which the parameter changes take place. One of the most popular ways of learning with the training set is the *batch mode* learning. This method consists on first average the learning rule over all P training examples before changing the weights and bias (Haykin, 1999; Heskes & Wiegerinck, 1996).

In fact, the *batch* method computes the weights and bias updates for each input sample and store these values (without changing the weights) during one pass through the training set, which is called an epoch. At the end of each epoch, all the weights/bias updates are added together, and only then will the weights be updated with the composite value. This method adapts the weights with a cumulative weights update (J. Principe et al., 2000).

Another learning strategy is the *incremental* learning. At this learning method a pattern is presented to the network and the weights are updated before next pattern is considered. *On-line* learning is a type of *incremental* strategy (Heskes & Wiegerinck, 1996). *Batch-mode* learning is completely deterministic while *incremental* learning strategies are stochastic due to the arbitrariness on the order by which the patterns are presented. The *batch-mode* requires additional storage for each weight which implies a heavier demand of memory in comparison with the *incremental mode*.

42 5 Gradient Method

The problem of learning in NN has been formulated in terms of the optimization of a cost function/performance index F. This cost function/performance index is a function of the adaptive parameters (synaptic weights and biases) in the network. The problem of optimizing functions of many variables has been widely studied, and is directly applicable to the training of neural networks. Several of the most important practical algorithms to train NN can be found in (Bishop, 1995) and (Hagan, Demuth, & Beale, 1996) (Chapter 7 and Chapter 9, respectively). One of the simplest of these is Gradient Method (GM), sometimes also known as *Steepest Descent* (for the minimization context) or as *Steepest Ascent* (for the maximization context) (Miranda, 2007). The GM is an iterative algorithm. Giving a differentiable function $F(\mathbf{x})$, with n-dim vector $\mathbf{x}$, the iterative process starts from some initial guess, $\mathbf{x}^0$, and then updates the guess value in stages according to:

$$\mathbf{x}^{new} = \mathbf{x}^{old} + \eta \, \nabla F \qquad \text{eq. R4/(4)}$$

This means that each component x_k^{new} comes:

$$x_k^{new} = x_k^{old} + \eta \left. \frac{\partial F}{\partial x_k} \right|_{x_k = x_k^{old}} \quad \text{eq. R4/(5)}$$

The parameter η corresponds to the iteration step and will be positive for maximizing and negative for minimizing. Gradient Descent can be seen in more detail at (Bishop, 1995) (Chapter 7, Section 7.5).

43 6 Supervised and Unsupervised Training

One of the classifications in NN is the one according to the teacher/target existence in learning process. In that way, NN can be classified in *supervised* (with teacher) or *unsupervised* (without teacher) (Haykin, 1999). In *supervised* learning, a desired response (target value) is available indicating, e.g., the amount of error (difference between the desired response and the actual response) in system performance. This is in contrast with *unsupervised* learning, where no teacher is available (Kishan, Mohan, & Ranka, 1996), i.e., there is no *a priori* output. To perform *unsupervised* learning a competitive learning rule may be use. In fact, different algorithms for *unsupervised* learning rule can be seen in (Haykin, 1999) (Chapter 8).

44 7 Train, Test and Validation Set

The data set is usually divided in two (train and test) or three (train, test and validation) sets. This work considered a division in three parts: train, test and validation. The training dataset is applied to build up the model adjusting the weights of the NN, the test dataset is applied during the NN training in order to prevent overfitting, finally, the validation dataset is applied after the train of the NN to validate the model built and to determine how well the predictive model generalizes.

The number of examples to include in each dataset is not consensual in literature. Some dataset divisions suggested in literature are i) 50% + 25% + 25%; ii) 70% + 10% + 20% or iii) 40% + 30% + 30%. This work adopted the division 50% + 25% + 25%, with 1000 (50%), 500 (25%) and 500 (25%) examples.

Another important topic is to ensure that the datasets of train, test and validation belong to the same population. A statistical analysis applying the statistical tests Smirnov and Cramer-Von-Mises was applied, as detailed in (Hora & Palma, 2012).

45 8 Size of Training Set

Another topic, related with above section, is how to fix a good size, N, for the training set. Again no fixed rule exists for this question. In this case, a rule of thumb can be used and states that $N \approx 10 * weights$ (J. Principe et al., 2000, p. 199). In our case, with a network $[24 \ - \ 12 \ - \ 24]$, we have:

$$N \approx 10 * weights \qquad \text{eq. R4/(6)}$$

$$\Leftrightarrow N \approx 10 * (2 * 24 * 12) \qquad \text{eq. R4/(7)}$$

$$\Leftrightarrow N \approx 5760 \qquad \text{eq. R4/(8)}$$

However, in this work our available data consist on only 1000 patterns.

46 9 Data Normalization

Normalization is a NN pre-processing that as a high importance on training NN to obtain good results and reduce significantly the calculation time (Sola & Sevilla, 1997). Normalization is often useful if different variables have typical values which are on widely different scales/differ by several orders of magnitude. Applying a linear transform, inputs are arranged to have similar values. This transformation can be applied to each variable independently (Bishop, 1995; Jayalakshmi & Santhakumaran, 2011) - Normalization by Entrance or can be applied considering all data - Global Normalization. At the same time, there are different techniques to perform data normalization (Bishop, 1995; Jayalakshmi & Santhakumaran, 2011; Kim, 1999). The ones chosen for this work were the MinMax and Z-Score Normalization.

MinMax Normalization

$$\tilde{x}_i^n = \frac{x_i^n - min_{i,o}}{max_{i,o} - min_{i,o}} \cdot \left(max_{i,n} - min_{i,n}\right) + min_{i,n} \qquad \text{eq. R4/(9)}$$

Where x_i^n and $\tilde{x}_i^n$ are the raw input data and the input data normalized, respectively ; $max_{i,o}$and $min_{i,o}$ are the maximum and minimum for the original input variable i (**Entrance**), respectively; $max_{i,n}$and $min_{i,n}$ are the new range of input variable i. As hyperbolic tangent (*tanh*) is the chosen activation function than $max_{i,n} = 1$ and $min_{i,n} = -1$ (Kim, 1999) and eq. R4/(9) becomes:

$$\tilde{x}_i^n = \frac{2 \cdot x_i^n - max_{i,o} - min_{i,o}}{max_{i,o} - min_{i,o}} \qquad \text{eq. R4/(10)}$$

A similar result arises to **global** normalization:

$$\tilde{x}_i^n = \frac{2 \cdot x_i^n - max_o - min_o}{max_o - min_o} \qquad \textbf{eq. R4/(11)}$$

Where max_o and min_o are the global maximum and minimum for original data.

This type of normalization was chosen instead other (e.g., Z-Score) because data set is a regular set without outliers.

Statistical or Z-Score Normalization

As input variables, x_i^n from training set, are treat independently (Entrance), the variance and mean calculation are done for each of them:

$$\bar{x}_i = \frac{1}{N}\sum_{i=1}^{N} x_i^n \qquad \textbf{eq. R4/(12)}$$

$$\sigma_i^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i^n - \bar{x}_i)^2 \qquad \textbf{eq. R4/(13)}$$

Where $n = 1, \ldots, N$ labels the instances. Using these statistical concepts, a re-scaled set of variables are define as (Bishop, 1995):

$$\tilde{x}_i^n = \frac{x_i^n - \bar{x}_i}{\sigma_i} \qquad \textbf{eq. R4/(14)}$$

A similar result derives for global issue computing the mean and standard deviation over all train set.

The advantage of this type of norm (Z-Score) is that it reduces the effects of outliers in the data (Bishop, 1995; El-Sharkawi, 1995; Jayalakshmi & Santhakumaran, 2011).

47 10 Number of Hidden Neurons (NHN)

The input and output number of neurons are determined by the number of available inputs and required outputs respectively. The only thing remaining is how to determine the number of neurons in the hidden layer. It is important to note that using too few hidden neurons will result in under fitting while too many hidden neurons may result in over fitting (Panchal, Ganatra, Kosta, & Panchal, 2011; J. Principe et al., 2000, p. 143). In fact, the best number of hidden units depends on many factors, e.g., the number of input and

output units, the number of training cases, the complexity of cost function, the type of activation function, the training algorithm, among other (Panchal et al., 2011). Some books and articles[1] offer *rules of thumb* for choosing an architecture, however these rules do not consider, e.g., the number of training cases.

An interesting behaviour according to the NHN was shown by El-Sharkawi (El-Sharkawi, 1995) in next figure, which shows the comparative cross validation among two or more NN as a method of determining the NHN. El-Sharkawi concludes that the best range for NN should be chosen in range where the error of the NN is relatively unchanged. Ultimately, the selection of NHN can be done by trial and error (Panchal et al., 2011).

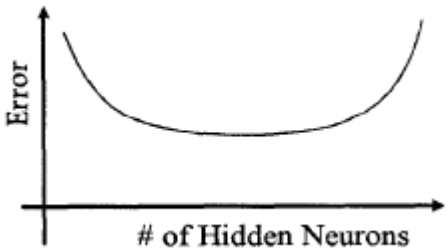


Figure R4/ 3 - NN Error versus NHN, from (El-Sharkawi, 1995).

48 11 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The new components are chosen in such way that the first new component has maximum variance, the second new component has maximum variance subject to being uncorrelated with the first one, and so forth. PCA can be done by eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix, usually after normalizing the data matrix (Mohamad-Saleh & Hoyle, 2008).

[1] NOTE: Bibliography extracted from http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html

- Blum, A. (1992), Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems, NY: Wiley, p. 60.
- Swingler, K. (1996), Applying Neural Networks: A Practical Guide, London: Academic Press, p. 53.
- Berry, M.J.A., and Linoff, G. (1997), Data Mining Techniques, NY: John Wiley & Sons, p. 323.
- Boger, Z., and Guterman, H. (1997), "Knowledge extraction from artificial neural network models," IEEE Systems, Man, and Cybernetics Conference, Orlando, FL.

In that way, PCA technique projects the data into a linear subspace with intended minimum information loss, by multiplying the data by the eigenvectors of the sample covariance matrix. A point is then represented by its coordinates along the directions of greatest variance in the data set (Costa, 2008).

To our work, as we have a fixed number of hidden neurons (13 neurons), the thirteen eigenvectors corresponding to the highest eigenvalues were chosen. In fact, we are able to chosen at maximum *DimIn* eigenvectors. At code we use a function from Armadillo (C++ linear algebra library): *Princomp* function - that performs a principal component analysis of matrix X, where each row of X is an observation and each column is a variable. The computation is based on singular value decomposition.

First Half of AA: Weights and Bias Initialization

Bias and *weights* initialization can be done by two different processes. For *weights* these two processes change between *random* or *PCA*. And for *Bias* the processes are: *random* or *zeros*. If *PCA* is chosen it is natural that for *Bias* the *zeros* initialization is chosen instead *random* option. In fact, as *PCA* is a good optimal start point, should not be added more options, what leads to choose a *zeros Bias* initialization. So, this should be the optimal starting point for *Bias* when *PCA* is preferred.

Second Half of AA: Weights and Bias Initialization

As the first half of the neural network approximates the function f and the second half approximates the inverse function f^{-1}, it is natural that a good initialization for weights of second half of autoenconder be the transpose of final weights obtained for the first half.

$$W_{2nd\ half} = \left(W_{1st\ half}\right)^T \qquad \textbf{eq. R4/(15)}$$

The method for the *Bias* initialization of second part can be chosen between *random* or *zeros*.

49 12 Stability - Oja's Rule (Weights Normalization)

Oja's learning rule (named from Erkki Oja) is a model of how neurons in the brain or in artificial neural networks change connection strength, or learn, over time. Hebbian update rule is unstable since the weights grow without bound (J. Principe et al., 2000). Oja's Rule is a modification of the standard Hebb's Rule that, through multiplicative weights normalization, solves all stability problems. Hebbian learning updates the weights according to:

$$w(n+1) = w(n) + \eta\, x(n) y(n) \qquad \textbf{eq. R4/(16)}$$

Where n is the iteration number and η a step size. Considering the simplest normalization, proposed by Oja (Oja, 1982), the new value of the weight become:

$$w_i(n+1) = \frac{w_i(n) + \eta\, y(n) x_i(n)}{\sqrt{\sum_i (w_i(n) + \eta\, y(n) x_i(n))^2}} \qquad \textbf{eq. R4/(17)}$$

Oja proposed an approximated update of (Oja, 1982):

$$w_i(n+1) = w_i(n) + \eta\, y(n)\big(x_i(n) - y(n) w_i(n)\big) \qquad \textbf{eq. R4/(18)}$$

$$= w_i(n)[1 - \eta\, y^2(n)] + \eta\, x_i(n)\, y(n)$$

producing the Oja's rule.

One of the objectives of our work included the networks competition: two networks were trained, one with train data set for type 1 and other with train data from type 2. Then, the objective is passing through two networks the validation set with general data from type 1 and type 2. The network that gives the lowest error for each example decides the type of that example. For a fair competition, the errors should correspond to the same magnitude. Normalization from Oja's Rule provides a way to achieve this. Our code implements this normalization proposed by Oja, while the learning process is occurring. The weights normalization is done by column, includes the bias normalization and is processed for the first and second half of AA.

Relating Oja's Rule with PCA

Oja's Rule derived as a limit process from an earlier well-known formulation of the Hebbian-type modification, leads to a behaviour where the unit is able to extract from its input a statistical technique known as principal component analysis (Oja, 1982). A detailed explanation on the correspondence between Oja's Rule and PCA is provided by (J. Principe et al., 2000, p. 294): "Hebbian learning finds the direction where the input data has the largest projection. But the weight vectors grow without limit. With Oja's rule we found a way to normalize the weight vector to 1. We should expect that this normalization would not change the geometric picture developed for the Hebbian network. In fact, it is possible to conclude that training the linear processing element (neuron) with Oja's algorithm produces a weight vector that is the eigenvector of the input autocorrelation matrix, and produces at its output the largest eigenvalue."

50 13 Activation function

Activation function, $\varphi(\cdot)$ at Equation **Error! Reference source not found.**, also called Transfer Function, is used for limiting the amplitude of the output of a neuron (typically $[0, 1]$ or $[-1, 1]$). Linear, Piecewise-Linear (Haykin, 1999, pp. 35-36) / Symmetric Saturating Linear (Hagan et al., 1996), Logistic and Hyperbolic Tangent functions are some examples of activation functions. More functions can be explored at (Hagan et al., 1996, pp. 2-6). Next equations refer to those activations functions that were implemented in our code and their derivatives (Haykin, 1999, p. 190):

Linear

The simple linear function is defined as:

$$\varphi(u) = u \qquad \textbf{eq. R4/(19)}$$

Figure R4/ 3 illustrates this function:

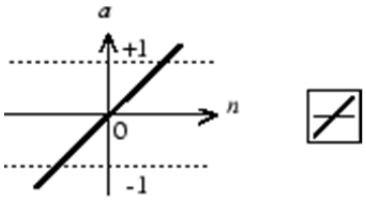


Figure R4/ 4 - Linear Transfer Function, from (Demuth & Beale, 2002).

The correspondent derivative is:

$$\varphi'(u) = 1 \qquad \textbf{eq. R4/(20)}$$

Symmetric Saturating Linear

Next equation corresponds to the symmetric saturating linear function (or Piecewise-Linear), described in Figure R4/ 5:

$$\varphi(u) = \begin{cases} -1, & u < -1 \\ u, & -1 \leq u \leq 1 \\ 1, & u > 1 \end{cases} \qquad \textbf{eq. R4/(21)}$$

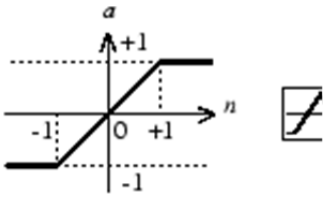


Figure R4/ 5 - Symmetric Saturating Linear Transfer Function, from (Demuth & Beale, 2002).

This function avoids saturation unlike simple linear function. The correspondent derivative of this function (Demuth & Beale, 2002) is:

$$\varphi'(u) = \begin{cases} 1, & -1 \leq u \leq 1 \\ 0, & otherwise \end{cases}$$ **eq. R4/(22)**

Logistic

Logistic sigmoid transfer function is defined as:

$$\varphi(u) = \frac{1}{1 + e^{-\frac{u}{a}}}$$ **eq. R4/(23)**

Where a is the activation function flatness that defines the slope of this function. Next figure describes this function:

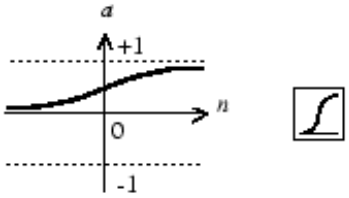


Figure R4/ 6 - Logistic Sigmoid Transfer Function, from (Demuth & Beale, 2002).

The correspondent derivative is:

$$\varphi'(u) = \frac{1}{a} \cdot \frac{e^{-\frac{u}{a}}}{\left(1 + e^{-\frac{u}{a}}\right)^2}$$ **eq. R4/(24)**

Let be $y = \varphi(u)$. Then (Haykin, 1999, p. 190),

$$\varphi'(u) = \frac{1}{a} \cdot y(1 - y)$$ **eq. R4/(25)**

This equality derives from:

$$y(1 - y) = \varphi(u)\big(1 - \varphi(u)\big)$$

$$= \frac{1}{1+e^{-\frac{u}{a}}}\left(1 - \frac{1}{1+e^{-\frac{u}{a}}}\right)$$ **eq. R4/(26)**

$$= \frac{1+e^{-\frac{u}{a}}-1}{\left(1+e^{-\frac{u}{a}}\right)^2}$$

$$= \frac{e^{-\frac{u}{a}}}{\left(1+e^{-\frac{u}{a}}\right)^2}$$

Resuming,

$$\varphi'(u) = \frac{1}{a} \cdot \varphi(u)\big(1 - \varphi(u)\big) \qquad \textbf{eq. R4/(27)}$$

Hyperbolic Tangent

The Hyperbolic Tangent function can be defined by:

$$\varphi(u) = \frac{2}{1 + e^{-\frac{u}{a}}} - 1 \qquad \textbf{eq. R4/(28)}$$

or

$$\varphi(u) = \frac{e^{u/a} - e^{-u/a}}{e^{u/a} + e^{-u/a}} = tanh\left(\frac{u}{a}\right) \qquad \textbf{eq. R4/(29)}$$

The slope of this function is defined by a, called activation function flatness. Next figure illustrates the function.

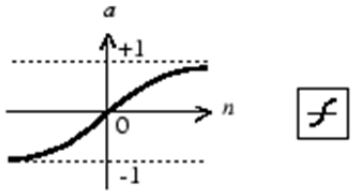


Figure R4/ 7 - Hyperbolic Tangent Function, from (Demuth & Beale, 2002).

Proof of equivalence between eq. R4/(28) and (29):

$$tanh(u) = \frac{\sinh(u)}{\cosh(u)} \qquad \textbf{eq. R4/(30)}$$

$$= \frac{\frac{e^u - e^{-u}}{2}}{\frac{e^u + e^{-u}}{2}}$$

$$= \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$

$$= \frac{1 - e^{-2u}}{1 + e^{-2u}}$$

$$= \frac{2 - 1 - e^{-2u}}{1 + e^{-2u}}$$

$$= \frac{2}{1 + e^{-2u}} - 1$$

So,

$$\varphi(u) = tanh\left(\frac{u}{2a}\right) \qquad \textbf{eq. R4/(31)}$$

The correspondent derivative is:

$$\varphi'(u) = \left(tanh\left(\frac{u}{2a}\right)\right)'$$

$$= \frac{1}{2a} \cdot \left(1 - tanh^2\left(\frac{u}{2a}\right)\right) \qquad \textbf{eq. R4/(32)}$$

$$= \frac{1}{2a} \cdot \left(1 - \varphi^2(u)\right)$$

Remember that,

$$\frac{d}{dx}\tanh(x) = \frac{d}{dx}\frac{senh(x)}{\cosh(x)}$$

$$= \frac{senh'(x)\cosh(x) - senh(x)\cosh'(x)}{\cosh^2(x)}$$

$$= \frac{\cosh^2(x) - senh^2(x)}{\cosh^2(x)} \qquad \textbf{eq. R4/(33)}$$

$$= 1 - \frac{senh^2(x)}{\cosh^2(x)}$$

$$= 1 - tanh^2(x)$$

51 14 Stable Learning Rate

Learning rate is the parameter η at Equation **Error! Reference source not found.**. When this parameter is large learning occurs quickly, but if it is too large it may lead to instability and errors may even increase (Demuth & Beale, 2002). To ensure stable learning, the learning rate coefficient should respect a threshold. Next Equation **Error! Reference source not found.** states the maximum allowed stable learning rate for quadratic functions as inversely proportional to the largest eigenvalue, λ_{max}, of Hessian matrix (Hagan et al., 1996):

$$\eta \leq \frac{2}{\lambda_{max}} \qquad \textbf{eq. R4/(34)}$$

A detailed overview on this topic is provided by (Hagan et al., 1996, p. 9.6).

Hagan (Hagan et al., 1996, p. 10.10) and Principe (J. C. Principe, 2010, p. 27) provide an equivalent result (eq. R4/35)) for LMS algorithm, but now using the eigenvalues, λ'_{max}, of the input correlation matrix (recall that $HessianMatrix = 2 * CorrelationMatrix$).

$$\eta \leq \frac{1}{\lambda'_{max}} \qquad \textbf{eq. R4/(35)}$$

Other way to fix the maximum stable learning rate was provided by Principe (meeting at 04/07/2012). According to him, to ensure stable learning, the learning rate must be less than the reciprocal of two times the largest eigenvalue of the correlation matrix of the input vectors (eq. R4/(36).

$$\eta \leq \frac{1}{2 * \lambda'_{max}} \qquad \textbf{eq. R4/(36)}$$

Maximum stable learning rate was implemented in our code according to this equation.

52 15 Adaptive Learning Rate

There are two general methods to determine the learning coefficient: minimize the cost function performance with respect to learning rate at each iteration or predetermine a value for the learning rate. It is not practical to determine the optimal setting for the learning rate before training, and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface. The performance of the steepest descent algorithm can be improved if we allow the learning rate to change during the training process. An adaptive learning rate will attempt to keep the learning step size

as large as possible while keeping learning stable (Demuth & Beale, 2002). According to (Haykin, 1999, p. 213), an adaptive learning rate was implemented in all autoencoders. The rules used to define the adaptive learning evolution are the ones define in (Hagan et al., 1996, p. 12.12).

53 16 Parzen Window Width - Silverman Rule

Information Theory concepts are based on probability density functions that are often hard to obtain. The Parzen window method is a non-parametric method to estimate these probability density functions. The method involves placing a kernel function on top of each sample and evaluating the density as a sum of kernels. A Gaussian kernel in d-dimensional space is defined as (Torkkola, 2003):

$$G(x,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\ |\Sigma|^{\frac{1}{2}}} \cdot exp\left\{-\frac{1}{2}x^T\Sigma^{-1}x\right\} \qquad \textbf{eq. R4/(37)}$$

Chose an appropriate window size for eq. R4/(37) is not easy (Jenssen, Principe, Erdogmus, & Eltoft, 2006). However the Silverman's Rule gives us optimal window size estimation (Silverman, 1986, p. 87), for the multivariate normal case:

$$\sigma_{opt} = \hat{\sigma} \cdot h_{opt}$$

$$= \hat{\sigma} \cdot A(k) \cdot n^{-\frac{1}{(d+4)}} \qquad \textbf{eq. R4/(38)}$$

$$= \hat{\sigma} \cdot \left\{\frac{4}{(d+2)}\right\}^{\frac{1}{(d+4)}} \cdot n^{-\frac{1}{(d+4)}}$$

Where $\hat{\sigma}^2$ is the average marginal variance:

$$\hat{\sigma}^2 = d^{-1} \cdot \sum_i \sigma_{ii}^2 \qquad \textbf{eq. R4/(39)}$$

d is the dimension, in our case is the hidden vector dimension and n is the size of train set. In our code we used as a base value this optimal estimation of σ. Other values, based on a percentage of this value, were explored.

54 17 Adaptive Sigma, σ (2nd half of AA)

In first half of autoencoder, the σ parameter was calculated based on Silverman's Rule and set unmodified over all epochs, as the learning process following ITL concepts is a heavy process. The training of second half can be chosen between *propagation*, *CIM* (correntropy induced metric) and *Corr* (only correntropy). To implement the last two options a kernel width parameter, σ, should be defined. For this, code implementation comprises an adaptive σ algorithm according to the update rule (Singh & Principe, 2010):

$$\sigma_{n+1} = \sigma_n + \eta \left[\frac{\sum_{i=n-L}^{n-1} exp\left(-\frac{(e_n - e_i)^2}{2\sigma_n^2}\right) \cdot \left(\frac{(e_n - e_i)^2}{\sigma_n^3} - \frac{1}{\sigma_n}\right)}{\sum_{i=n-L}^{n-1} exp\left(-\frac{(e_n - e_i)^2}{2\sigma_n^2}\right)} \right] \quad \textbf{eq. R4/(40)}$$

Where σ_{n+1} is the new value of σ; σ_n is the old value of σ, given by Silverman's rule; η is the learning step set equal to 0.5; L is the window size, i.e., defines how many old σ values influence the calculation of actual σ and e_i is the component i of error vector, of dimension $L + 1$, that holds the L oldest errors plus the actual one. L was set equal to 4. A detailed description of the training methods referred above is given at report (Palma & Hora, 2012).

55 18 Saturation

The saturation problem appears when the nonlinear activation functions reach its upper or lower saturation limits. As El-Sharkawi explains (El-Sharkawi, 1995), any wide change in the input would produce no or minimal change in the output and the neurons in this case are paralyzed. So it is common and acceptable to have some neurons in the saturation region, but too many would render the neural network useless. If network reaches saturation, the neurons must be randomly perturbed and the learning process continued.

Our implementation includes a saturation verification block. When a specific number of neurons (30%) reach the saturation limit (set equal to 0.9, this means, the neurons with an output greater or equal than 0.9 are considered a saturated ones) the algorithm randomly perturbed the weights and bias.

56 19 Overfitting

According to (El-Sharkawi, 1995) there is a difference between training and memorization. A model is typically trained by maximizing its performance on some set of training data. By the other hand, the model

efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data.

When a model begins to memorize training data rather than learning to generalize from trend this is called overfitting. Overfitting corresponds a test data error much higher that the train data error and means that the neural system is over determined. A properly trained system should respond with same error measures to both training and testing data. To avoid overfitting, this point must be identified and the training must be stoped. A detailed explanation is given at (El-Sharkawi, 1995). The figure illustrates the point where the optimal learning and generalization are achieved, that is close to the global minimum of test error:

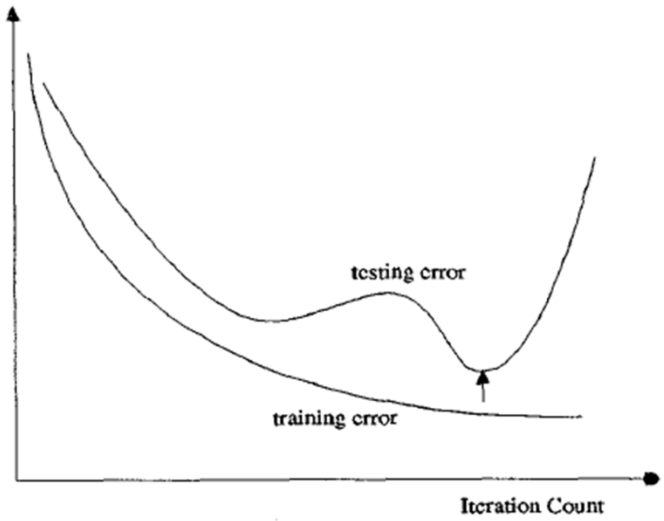


Figure R4/ 8 - Overfitting, from (El-Sharkawi, 1995).

57 20 Concluding Remarks

This work synthesizes theoretical aspects which are relevant to consider during the implementation of neural networks. For each topic addressed the most relevant information was detailed.

58 Bibliography

Bishop, C. (1995). *Neural Networks for Pattern Recognition*: Clarendon Press Oxford.

Costa, L. (2008). *Application of Evolutionary Swarms and Autoencoders to Wind-Hydro coordination.* Faculty of Engineering of the University of Porto, Porto, Portugal.

Demuth, H., & Beale, M. (2002). Neural Network Toolbox - User's Guide (Version 4). *The MathWorks*. Retrieved from The MathWorks website: http://cda.psych.uiuc.edu/matlab_pdf/nnet.pdf

El-Sharkawi, M. A. (1995). *Neural Network Application to High Performance Electric Drives Systems.* Paper presented at the Proceedings of the 1995 IEEE IECON 21st International Conference on Industrial Electronics, Control, and Instrumentation.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design*. Boston and London: Pws Pub.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation* (2nd ed.). Ontario, Canada: Pearson Education.

Heskes, T., & Wiegerinck, W. (1996). A Theoretical Comparison of Batch-Mode, on-Line, Cyclic, and Almost Cyclic Learning. *IEEE Transactions on Neural Networks, 7*(4), 919-925.

Hora, J., & Palma, V. (2012). Analysis using descriptive statistics on the data used for the ITL networks and on the data used in the Topology problem (Power System) *INESC Interim Reports*. Porto, Portugal: INESC TEC.

Jayalakshmi, T., & Santhakumaran, A. (2011). Statistical Normalization and Back Propagation for Classification. *International Journal of Computer Theory and Engineering, 3*(1), 1793-8201.

Jenssen, R., Principe, J., Erdogmus, D., & Eltoft, T. (2006). The Cauchy–Schwarz divergence and Parzen windowing: Connections to graph theory and Mercer kernels. *Journal of the Franklin Institute, 343*(6), 614-629.

Kim, D. (1999). Normalization Methods for Input and Output Vectors in Backpropagation Neural Networks. *International Journal of Computer Mathematics, 71*(2), 161-171.

Kishan, M., Mohan, C., & Ranka, S. (1996). Elements of Artificial Neural Networks (Complex Adaptive Systems): The MIT Press.

Miranda, V. (2007). Redes Neuronais – Treino por Retropropagação (Texto de apoio à disciplina de Controlo Difuso e Redes Neuronais do 5º ano da LEEC). Porto, Portugal.

Miranda, V., Krstulovic, J., Keko, H., Moreira, C., & Pereira, J. (2012). Reconstructing Missing Data in State Estimation With Autoencoders. *IEEE Transactions on Power Systems, 27*(2), 604-611.

Mohamad-Saleh, J., & Hoyle, B. S. (2008). Improved Neural Network Performance Using Principal Component Analysis on Matlab. *International Journal of The Computer, the Internet and Management, 16*(2), 1-8.

Oja, E. (1982). A Simplified Neuron Model as a Principal Component Analyser. *Journal of Mathematical Biology, 15*(3), 267-273.

Palma, V., & Hora, J. (2012). Theoretical Concepts of ITL Neural Networks *INESC Interim Report*. Porto, Portugal.

Panchal, D., Ganatra, A., Kosta, Y. P., & Panchal, D. (2011). Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers. *International Journal of Computer Theory and Engineering, 3*(2), 332-337.

Principe, J., Euliano, N., & Lefebvre, W. (2000). *Neural and Adaptive Systems - Fundamentals Through Simulations* (1st ed.): John Wiley & Sons, Inc.

Principe, J. C. (2010). Information Theoretic Learning Renyi's Entropy and Kernel Perspectives: Springer.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. London, UK: Chapman & Hall/CRC.

Singh, A., & Principe, J. (2010). *Kernel width adaptation in information theoretic cost functions.* Paper presented at the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP).

Sola, J., & Sevilla, J. (1997). Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems. *IEEE Transactions on Nuclear Science, 44*(3), 1464-1468.

Torkkola, K. (2003). Feature extraction by non parametric mutual information maximization. *Journal of Machine Learning Research, 3*, 1415-1438.

Theoretical Concepts of ITL Neural Networks

PTDC/EEA-EEL/104278/2008

Report LASCA / R5

59 1 Introduction

The training of a neural network is performed with the optimization of a specified cost function F. This work applies the Steepest Descent Method (SDM) as the algorithm basing the optimization of neural networks. The SDM is an iterative process that allows a neural network to be improved towards an optimum, that can be a local or a global optimum. Equation R5/(1) shows the general adaptation of weights used within SDM.

$$\omega_{ij}{}^{k+1} = \omega_{ij}{}^{k} + \Delta\omega_{ij}$$

$$\omega_{ij}{}^{k+1} = \omega_{ij}{}^{k} + \eta \frac{\partial F}{\partial \omega_{ij}} \qquad \textbf{eq. R5/(1)}$$

The SDM performs adjustments in the neural network weights, as defined in the following expression:

$$\Delta\omega_{ij} = \eta \frac{\partial F}{\partial \omega_{ij}} \qquad \textbf{eq. R5/(2)}$$

The parameter η corresponds to the *iteration step* or *learning coefficient*. This parameter assumes positive values when the optimization problem is to be maximized, and it assumes negative values when the optimization is to be minimized. A momentum term may be added to the SDM algorithm, which is intended to prevent the optimization process to be trapped at local optima. Further reading on this subject can be found in (Miranda, 2007). The algorithm SDM can be generally defined as follows:

$$\Delta\omega_{ij}{}^{(t)} = \eta \frac{\partial F}{\partial \omega_{ij}} + \alpha \Delta\omega_{ij}{}^{(t-1)} \qquad \textbf{eq. R5/(3)}$$

60 2 Classic PROP theory overview

The back propagation method uses a supervised approach to train autoencoders. Accodingly, the train of an autoencoder is performed with the minimization of the Mean Squared Error (MSE). The MSE is defined in the next expression (where N is the total number of instances within the train set, and M is the total number of neurons within the input layer) (Beale, Hagan, & Demuth, 2012):

$$MSE = \frac{1}{N \times M} \sum_{p=1}^{N} \sum_{i=1}^{M} (e_{pi})^2 = \frac{1}{N \times M} \sum_{p=1}^{N} \sum_{i=1}^{M} (T_{pi} - O_{pi})^2 \qquad \text{eq. R5/(4)}$$

The derivative of the MSE in order to the output O_{ij} is defined as follows.

$$\frac{\partial}{\partial O_{pi}} MSE = -\frac{2}{N \times M} (T_{pi} - O_{pi}) \qquad \text{eq. R5/(5)}$$

The autoencoders considere in this work are trained to minimize the function MSE, using the algorithm SDM (Miranda, 2007). Moreover, the adaptive learning rate was applied to the classic PROP algorithm. Figure R5/ 1 provides the schematic visualization of this training procedure.

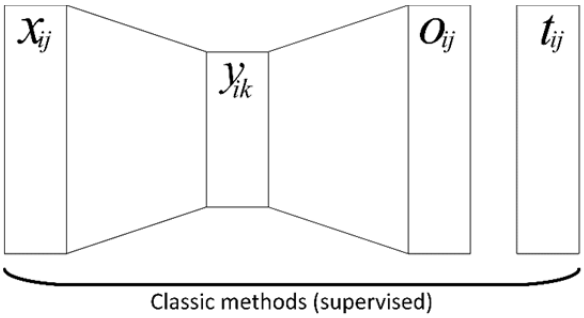


Figure R5/ 1 - Schematic of an autoencoder trained with backpropagation.

Where,

- a_{pi} = neuron i at input layer for instance p;
- h_{pj} = neuron j at hidden layer for instance p;
- o_{pi} = neuron i at output layer for instance p;
- $t_{pi} = y_{pi}$ target neuron i for instance p;

61 3 ITL Theory overview

This report includes insights on applying Information Theoretic Learning ITL criteria to train the 1st half of the autoencoder separately. The 2nd half of the autoencoder is trained in a subsequente stage with classic backpropagation algorithm, using the train dataset as the target ($t_{ij} = a_{ij}$). The ITL train of the 1st half of the autoencoder is unsupervised. In an unsupervised learning method (against a supervised approach), there is no formal desired response, the target. The representation of this approach is provided in Figure R5/ 2.

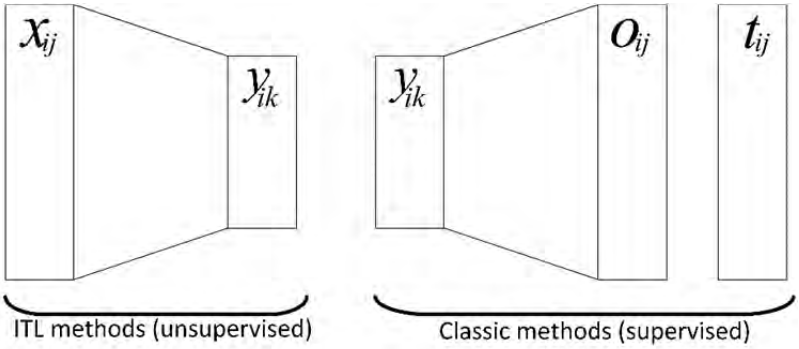


Figure R5/ 2 - Schematic of an autoencoder trained within two separated parts.

As explained in (Principe, n.d.), ITL algorithms are based on a combination of nonparametric estimations of probability density functions. Let $a_{p\cdot} \in R^M$ be the input instances (each instance with M neurons) and $h_{p\cdot} \in R^L$ be the hidden instance (with L neurons), with $p = 1, \dots, N$. From this point onwards, the notation used to refer to a general input instance is $\boldsymbol{y}$; to refer to a specific input instance p is $\boldsymbol{y}_p$ ($= \boldsymbol{a}_p = a_{p\cdot}$); to refer to a specific hidden instance p is $\boldsymbol{h}_p$ ($= h_{p\cdot}$). Then, the estimation of data PDF is given by the Parzen Window method using a Gaussian kernel:

$$\hat{f}_Y(\boldsymbol{y}) = \frac{1}{N}\sum_{p=1}^{N} G(\boldsymbol{y} - \boldsymbol{y}_p, \sigma^2 I) \qquad \textbf{eq. R5/(6)}$$

Where $\sigma^2 I$ is the covariance matrix; N is the number of instances ($TrainDim$); M is the number of neurons on input layer ($DimIn$); L is the number of neurons on hidden layer ($DimHidden$). The ITL methods proposed in this work are:

- Maximize the Entropy associated to the set of hidden instances, $\boldsymbol{h}_p, p = 1, \dots, N$;
- Maximize the Mutual Information between the set of input instances, $\boldsymbol{a}_p, p = 1, \dots, N$, and the set of hidden instances, $\boldsymbol{h}_p, p = 1, \dots, N$. This method was implemented using (i) Euclidean Distance and (ii) the Cauchy-Swartz Mutual Information estimators.

- Minimize the Mutual information between all different combinations of two neurons within the set of hidden instances (min $I_{CS}(h_{pi}, h_{pj})$: $\forall\, i \neq j\ ;\ i,j = 1, \dots, L\ ;\ p = 1, \dots, N$).

61.1 3.1 Entropy Maximization in the Hidden Layer (MaxE)

This unsupervised method aims to maximize the entropy of the hidden layer. The calculation of entropy is made using Renyi's Quadratic Entropy (this is the simpler form to calculate the derivation of the information forces (Principe, n.d.). This procedure has a complexity $O(N^2)$. The mathematic definition of Renyi's Quadratic Entropy is:

$$H_{R2}(Y) = -\log\left(\int_{-\infty}^{+\infty} {f_Y}^2(\boldsymbol{y})d\boldsymbol{y}\right) = -\log(V(Y)) \qquad \textbf{eq. R5/(7)}$$

Where $V(Y)$ is the Information Potential. An estimation for Information Potential is provided in equation R5/(8).

$$V(Y) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\int_{-\infty}^{+\infty} G(\boldsymbol{y} - \boldsymbol{y}_i, \sigma^2 I) G\left(\boldsymbol{y} - \boldsymbol{y}_j, \sigma^2 I\right) d\boldsymbol{y}$$
$$= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} G\left(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I\right) \qquad \textbf{eq. R5/(8)}$$

The physical interpretation is also provided:

$$V_{ij} = G\left(\boldsymbol{y}_i - \boldsymbol{y}_j, \sigma^2 I\right) = G\left(\boldsymbol{d}_{ij}, \sigma^2 I\right)$$

Where:

- y_i, y_j are the information particles; V_{ij} can be seen as the interactions between this particles.
- $V_i = \frac{1}{N}\sum_j V_{ij} = \frac{1}{N}\sum_j G\left(\boldsymbol{d}_{ij}, \sigma^2 I\right)$ is the sum of interactions on the $i - th$ information particle.
- $V = \frac{1}{N^2}\sum_i \sum_j V_{ij}$ is the sum of all pairs of interactions.=Information Potential

A physical interpretation of these quantities can be found in (Principe, n.d.). Maximize entropy of the hidden layer (MaxE) is equivalent to minimize the Information Potential $V(Y)$. The Information Potential was derivated (information forces) to perform the gradient method. Since derivative of a Gaussian is:

$$\frac{\partial}{\partial y_i} G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I) = -G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I) \cdot \frac{1}{2\sigma^2} \cdot (\boldsymbol{y}_i - \boldsymbol{y}_j) \quad \textbf{eq. R5/(9)}$$

The information forces become:

$$F_i = \frac{\partial}{\partial \boldsymbol{y}_i} V(Y) = -\frac{1}{N^2\sigma^2} \sum_{j=1}^{N} G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I)(\boldsymbol{y}_i - \boldsymbol{y}_j) = -\frac{1}{N^2\sigma^2} \sum_{j=1}^{N} V_{ij} \boldsymbol{d}_{ij} \quad \textbf{eq. R5/(10)}$$

The equations for the gradient method are:

$$\frac{\partial}{\partial \omega} V(Y) = \sum\nolimits_{i=1}^{N} \left[\frac{\partial}{\partial \boldsymbol{y}_i} V(Y)\right]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} = \sum\nolimits_{i=1}^{N} [F_i]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} \quad \textbf{eq. R5/(11)}$$

Where:

$$\frac{\partial \boldsymbol{y}_i}{\partial \omega} = \frac{\partial \boldsymbol{y}_i}{\partial net_i} \frac{\partial net_i}{\partial \omega} \quad \textbf{eq. R5/(12)}$$

And:

$$\boldsymbol{y}_i = \varphi(net_i) \quad \textbf{eq. R5/(13)}$$

φ is the activation function (see definition of net forward at eq. R5/(49)).

Cost Function

The cost function is the Entropy of the output vector:

$$H_{R2}(Y) = -\log(V(Y)) = -\log\left(\frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I)\right) \quad \textbf{eq. R5/(14)}$$

61.2 3.2 Mutual Information Methods

The idea of this method is use ITL to maximize the mutual information between input instances and the output at hidden layer, in sense to maximize the transfer information from data or minimize the mutual information between output neurons of the hidden layer to measure the independence between them. Mutual information can be computed as a difference of Shannon entropies:

$$I(X,Y) = H(X) - H(X|Y) \quad \text{eq. R5/(15)}$$

But as we seen above (see MaxE Chapter), Shannon entropy is not easy to estimate (Principe, n.d.). Principe suggests (Principe, 2010, n.d.)an alternative estimation for mutual information based on Kullback-Leibler divergence that estimates the divergence between the joint PDF and the factorized marginal.

$$I(X,Y) = KL\big(f_{XY}(x,y), f_X(x)f_Y(y)\big) = \int\int f_{XY}(x,y)\log\frac{f_{XY}(x,y)}{f_X(x)f_Y(y)}dxdy \quad \text{eq. R5/(16)}$$

But this is not quadratic in PDF so they cannot be easily integrated. Then Principe proposes new distances measures between two PDFs which contain only quadratic terms: Euclidean Distance ($\|x\|^2 + \|y\|^2 - 2x^Ty \geq 0$) and Cauchy-Schwartz inequality ($\log\frac{\|x\|^2\|y\|^2}{(x^Ty)^2} \geq 0$). The MI estimators become (Principe, n.d.):

$$I_{ED}(X,Y) = \int\int {f_{XY}}^2(x,y)dxdy - 2\int\int f_{XY}(x,y)f_X(x)f_Y(y)dxdy + \int\int {f_X}^2(x){f_Y}^2(y)dxdy \quad \text{eq. R5/(17)}$$

$$I_{CS}(X,Y) = \log\frac{\big(\int\int {f_{XY}}^2(x,y)dxdy\big)\big(\int\int {f_X}^2(x){f_Y}^2(y)dxdy\big)}{\big(\int\int f_{XY}(x,y)f_X(x)f_Y(y)dxdy\big)^2} \quad \text{eq. R5/(18)}$$

Defining the above terms like:

$$V_J = \int\int {f_{XY}}^2(x,y)dxdy$$

$$V_M = \int\int {f_X}^2(x){f_Y}^2(y)dxdy \quad \text{eq. R5/(19)}$$

$$V_C = \int\int f_{XY}(x,y)f_X(x)f_Y(y)dxdy$$

Corresponding to joint, marginal and cross information potentials, respectively. MI estimators can be rewritten as:

$$I_{ED}(X,Y) = V_J - 2V_C + V_M \quad \text{eq. R5/(20)}$$

$$I_{CS}(X,Y) = \log V_J - 2\log V_C + \log V_M \qquad \textbf{eq. R5/(21)}$$

Based on the Parzen Window method, the joint PDF and marginal PDF can be estimated as:

$$\hat{f}_{XY}(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N} G(\boldsymbol{x}-\boldsymbol{x}_i,\sigma^2 I)G(\boldsymbol{y}-\boldsymbol{y}_i,\sigma^2 I)$$

$$\hat{f}_X(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} G(\boldsymbol{x}-\boldsymbol{x}_i,\sigma^2 I) \qquad \textbf{eq. R5/(22)}$$

$$\hat{f}_Y(\boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N} G(\boldsymbol{y}-\boldsymbol{y}_i,\sigma^2 I)$$

And joint, marginal and cross information potentials become:

$$\hat{V}_J = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} G(\boldsymbol{x}_i-\boldsymbol{x}_j,2\sigma^2 I)G(\boldsymbol{y}_i-\boldsymbol{y}_j,2\sigma^2 I) \qquad \textbf{eq. R5/(23)}$$

$$\hat{V}_M = \hat{V}_x\hat{V}_y \quad \text{with} \quad \hat{V}_z = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} G\left(\boldsymbol{z}_i-\boldsymbol{z}_j,2\sigma^2 I\right) \qquad \boldsymbol{z}=\{\boldsymbol{x},\boldsymbol{y}\} \qquad \textbf{eq. R5/(24)}$$

$$\hat{V}_C = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{N}\sum_{j=1}^{N} G(\boldsymbol{x}_i-\boldsymbol{x}_j,2\sigma^2 I)\right)\left(\frac{1}{N}\sum_{j=1}^{N} G(\boldsymbol{y}_i-\boldsymbol{y}_j,2\sigma^2 I)\right) \qquad \textbf{eq. R5/(25)}$$

Consider, for the next section, a general variable $z=\{x,y\}$ and the next notation:

$$V_{ij}^{z} = G(\boldsymbol{z}_i-\boldsymbol{z}_j,2\sigma^2 I) \qquad \textbf{eq. R5/(26)}$$

$$V_i^z = \frac{1}{N}\sum_{j=1}^{N} V_{ij}^z = \frac{1}{N}\sum_{j=1}^{N} G(d_{ij},2\sigma^2 I) = \frac{1}{N}\sum_{j=1}^{N} G(z_i-z_j,2\sigma^2 I) \qquad \textbf{eq. R5/(27)}$$

$$V^z = \frac{1}{N}\sum_{i=1}^{N} V_i^z = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{N}\sum_{j=1}^{N} V_{ij}^z\right) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} V_{ij}^z \qquad \textbf{eq. R5/(28)}$$

Mutual Information Maximization between the input and hidden layer – Cauchy Schwartz Estimation (MaxMI_CS)

In this method, our aim is maximize MI between two vectors, vector of input instances and the vector of hidden layer outputs. The Cauchy Schwartz MI estimator between two vectors becomes (Principe, n.d.):

$$I_{CS}(X,Y) = \log \frac{V_J V_M}{(V_C)^2} \qquad \textbf{eq. R5/(29)}$$

Maximize Cauchy Schwartz MI is equivalent to maximize cross information potential (CIP), $V_{CIP} = \frac{V_J V_M}{(V_C)^2}$. In that case, the Information Potential derivative (information forces) to perform the SDM method in order to the variable $z = \{x, y\}$ becomes:

$$F_i^z = \frac{\partial}{\partial z} I_{CS}(X,Y) = \frac{\partial}{\partial z}\left(\log V_J - 2\log V_C + \log V_M\right)$$
$$= \frac{1}{V_J}\frac{\partial V_J}{\partial z} - 2\frac{1}{V_C}\frac{\partial V_C}{\partial z} + \frac{1}{V_M}\frac{\partial V_M}{\partial z} \qquad \textbf{eq. R5/(30)}$$

Where:

$$\frac{\partial V_J}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} G(\boldsymbol{x}_i - \boldsymbol{x}_j, 2\sigma^2 I) G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I)\right)$$
$$= -\frac{1}{N^2}\sum_{j=1}^{N} G(\boldsymbol{x}_i - \boldsymbol{x}_j, 2\sigma^2 I) G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I)\frac{\boldsymbol{z}_i - \boldsymbol{z}_j}{\sigma^2} \qquad \textbf{eq. R5/(31)}$$
$$= -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N} V_{ij}^x V_{ij}^y \boldsymbol{d}_{ij}$$

In our case, $\boldsymbol{z} = \{\boldsymbol{y}\}$ (neurons at hidden layer):

$$\frac{\partial V_J}{\partial \boldsymbol{y}} = -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N} V_{ij}^x V_{ij}^y \boldsymbol{d}_{ij}, \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j \qquad \textbf{eq. R5/(32)}$$

Similarly,

$$\frac{\partial V_C}{\partial \boldsymbol{y}} = -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N} V_j^x V_{ij}^y \boldsymbol{d}_{ij}, \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j \qquad \textbf{eq. R5/(33)}$$

$$\frac{\partial V_M}{\partial \boldsymbol{y}} = -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N} V_{ij}^y \boldsymbol{d}_{ij} \qquad \textbf{eq. R5/(34)}$$

And information forces are:

$$F_i = -\frac{1}{N^2\sigma^2}\left[\frac{\sum_j V_{ij}^x V_{ij}^y \boldsymbol{d}_{ij}}{\sum_i \sum_j V_{ij}^x V_{ij}^y} + \frac{\sum_j V_{ij}^y \boldsymbol{d}_{ij}}{\sum_i \sum_j V_{ij}^y} - 2\frac{\sum_j V_j^x V_{ij}^y \boldsymbol{d}_{ij}}{\sum_j V_j^x V_j^y}\right], \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j \qquad \textbf{eq. R5/(35)}$$

Remembering:

$$\frac{\partial}{\partial \omega} I_{CS}(X,Y) = \sum_{i=1}^{N}\left[\frac{\partial}{\partial \boldsymbol{y}_i} I_{CS}(X,Y)\right]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} = \sum_{i=1}^{N} [F_i]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} \qquad \textbf{eq. R5/(36)}$$

Cost Function

The cost function is the Cauchy Schwartz MI between inputs and hidden outputs:

$$I_{CS}(X,Y) = \log\frac{V_J V_M}{(V_C)^2} = \log\frac{\left(\frac{1}{N^2}\sum_i \sum_j V_{ij}^x V_{ij}^y\right)(V^x V^y)}{\left(\frac{1}{N}\sum_i V_i^x V_i^y\right)^2} \qquad \textbf{eq. R5/(37)}$$

Mutual Information Maximization between the input and hidden layer – Euclidean Distance Estimation (MaxMI_ED)

The Euclidean Distance MI estimator becomes (Principe, n.d.):

$$I_{ED}(X,Y) = V_J - 2V_C + V_M \qquad \textbf{eq. R5/(38)}$$

Maximize Euclidean Distance MI is equivalent to maximize cross information potential (CIP), $V_{CIP} = V_J - 2V_C + V_M$. The Information Potential derivative (information forces) to perform the SDM method in order to the variable $z = \{x, y\}$ becomes:

$$F_i^z = \frac{\partial}{\partial z} I_{ED}(X,Y) = \frac{\partial V_{ED}}{\partial z} = \frac{\partial}{\partial z}\left(V_J - 2V_C + V_M\right)$$

$$= \frac{\partial V_J}{\partial z} - 2\frac{\partial V_C}{\partial z} + \frac{\partial V_M}{\partial z}$$

eq. R5/(39)

In our case, $z = \{y\}$ (neurons at hidden layer):

$$F_i^y = \frac{\partial}{\partial y} I_{ED}(X,Y) = \frac{\partial V_{ED}}{\partial y}$$

$$= \frac{\partial}{\partial y}\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} V_{ij}^x V_{ij}^y - \frac{2}{N}\sum_{i=1}^{N} V_i^x V_i^y + V^x V^y\right)$$

$$= -\frac{1}{N^2}\sum_{j=1}^{N} V_{ij}^x \frac{1}{2\sigma^2} V_{ij}^y \, \boldsymbol{d}_{ij} - \frac{2}{N} V_i^x \left(-\frac{1}{N}\sum_{j=1}^{N}\frac{1}{2\sigma^2} V_{ij}^y \, \boldsymbol{d}_{ij}\right) + V^x \left(-\frac{1}{N^2}\sum_{j=1}^{N}\frac{-1}{2\sigma^2} V_{ij}^y \, \boldsymbol{d}_{ij}\right),$$

$$\boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j$$

$$= -\frac{1}{N^2 2\sigma^2}\left(\sum_{j=1}^{N} V_{ij}^x V_{ij}^y \, \boldsymbol{d}_{ij} - 2V_i^x \sum_{j=1}^{N} V_{ij}^y \, \boldsymbol{d}_{ij} + V^x \sum_{j=1}^{N} V_{ij}^y \, \boldsymbol{d}_{ij}\right), \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j$$

$$= -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N}\left(V_{ij}^x - 2V_i^x + V^x\right) V_{ij}^y \, \boldsymbol{d}_{ij}\,, \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j$$

$$= -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N} C_{ij}^x \; V_{ij}^y \; \boldsymbol{d}_{ij}\,, \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j \;\; and \; C_{ij}^x = V_{ij}^x - 2V_i^x + V^x$$

Concluding, information forces are:

$$F_i^y = -\frac{1}{N^2\sigma^2}\sum_{j=1}^{N} C_{ij}^x \; V_{ij}^y \; \boldsymbol{d}_{ij}$$

eq. R5/(40)

And SDM method becomes:

$$\frac{\partial}{\partial \omega} I_{ED}(X,Y) = \sum_{i=1}^{N}\left[\frac{\partial}{\partial \boldsymbol{y}_i} V_{ED}\right]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} = \sum_{i=1}^{N} [F_i]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega}$$

eq. R5/(41)

Cost Function

The cost function is the Euclidean Distance MI between inputs and hidden outputs:

$$I_{ED}(X,Y) = V_J - 2V_C + V_M$$

$$= \frac{1}{N^2}\sum_i\sum_j V_{ij}^x V_{ij}^y - \frac{2}{N}\sum_i V_i^x V_i^y + V^x V^y \qquad \text{eq. R5/(42)}$$

Or, according to (Xu & Principe, 1999):

$$I_{ED}(X,Y) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij}^x \; V_{ij}^y \qquad \text{eq. R5/(43)}$$

Mutual Information Minimization between all combinations of neurons of the hidden layer (Uni)

The method *MinHiddenNeuronMI_CS* minimizes the Mutual Information *I* between each combination of two different neurons integrating the hidden layer. This procedure ensures that the information contained in each hidden neuron is maximized, thus decreasing the replication of information over the hidden layer.

The considered pairs of different neurons are specified as (Y_1, Y_2), …, (Y_1, Y_L), …, (Y_k, Y_{k+1}), …, (Y_k, Y_L), …, (Y_{L-1}, Y_L), with $(k \neq L)$. One way to find the indexes of these combinations is to consider a squared matrix, with the position of the hidden layer neurons for both rows and columns, and chose the indexes related to the superior triangular positions of this matrix. A simple example is provided in Figure R5/ 3. Considering only the combinations of different neurons is adequate due to the mutual information property of symmetry $I(X,Y) = I(Y,X)$.

	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12
Y1		Y1, Y2	Y1, Y3	Y1, Y4	Y1, Y5	Y1, Y6	Y1, Y7	Y1, Y8	Y1, Y9	Y1, Y10	Y1, Y11	Y1, Y12
Y2			Y2, Y3	Y2, Y4	Y2, Y5	Y2, Y6	Y2, Y7	Y2, Y8	Y2, Y9	Y2, Y10	Y2, Y11	Y2, Y12
Y3				Y3, Y4	Y3, Y5	Y3, Y6	Y3, Y7	Y3, Y8	Y3, Y9	Y3, Y10	Y3, Y11	Y3, Y12
Y4					Y4, Y5	Y4, Y6	Y4, Y7	Y4, Y8	Y4, Y9	Y4, Y10	Y4, Y11	Y4, Y12
Y5						Y5, Y6	Y5, Y7	Y5, Y8	Y5, Y9	Y5, Y10	Y5, Y11	Y5, Y12
Y6							Y6, Y7	Y6, Y8	Y6, Y9	Y6, Y10	Y6, Y11	Y6, Y12
Y7								Y7, Y8	Y7, Y9	Y7, Y10	Y7, Y11	Y7, Y12
Y8									Y8, Y9	Y8, Y10	Y8, Y11	Y8, Y12
Y9										Y9, Y10	Y9, Y11	Y9, Y12
Y10											Y10, Y11	Y10, Y12
Y11												Y11, Y12
Y12												

Figure R5/ 3 - Illustration of the considered pairs of neurons within hidden layer.

The error of these combinations is only propagated to the respective weights. For example, the error associated with the combination (Y_1, Y_L) is only propagated to the input synapses in these two neurons. The Cauchy Schwartz mutual information theory described for the method *MaxMI_CS* is applied here, for each pair of neurons, and consequently following a one-dimensional approach.

Cost Function

The cost function of this method is the sum of the Cauchy Schwartz MI obtained for all different pairs of hidden neurons. This cost function is used following a minimization approach.

$$CostFunction = \sum_{n=1}^{L} \sum_{m=n+1}^{L} I_{CS}(Y_n, Y_m)$$

$$= \sum_{i=1}^{TOTALcomb} I_{CS}(comb_i), \qquad TOTALcomb = \frac{L}{2}(L-1)$$

eq. R5/(44)

62 4 Other Theory Overview

62.1 4.1 MinMax Normalization Method

The training of neural networks must include a pre-processing normalization of data. The advantage of this procedure is to remove the sacle dependencies of data. To perform this normalization we choose the Min-Max method:

$$data_{normalized} = \frac{data - min_{Orig}}{max_{Orig} - min_{Orig}}(max_{New} - min_{New}) + min_{New}$$

eq. R5/(45)

Where max_{New} is the maximum value for the normalized data; min_{New} is the minimum value for the normalized data; max_{Orig} is the maximum value for the original data; min_{Orig} is the minimum value for the original data. In our case, $max_{New} = 1$ and $min_{New} = -1$:

$$data_{normalized} = \frac{2 * data - max_{Orig} - min_{Orig}}{max_{Orig} - min_{Orig}}$$

eq. R5/(46)

There are other methods for data normalization but this method has the advantage of preserving exactly all relationships in the data, however it is sensitive to outliers. Another point in normalization procedure is the choose of max_{Orig} and min_{Orig}. In our code we implemented the methods:

- Global normalization: the chosen of max_a and min_a parameters can be done in all set of data (if they are similar, as our case);
- Normalization by entry: the chosen can be done by input matrix entry, this means that max_{Orig} and min_{Orig} choice need to be done for each input neuron in the correspondent set of instances.

Auxiliary calculations: aim of normalizing data process is transform the data in the original range $[min_{Orig}, max_{Orig}]$ into a new range $[min_{New}, max_{New}]$, as schematized in Figure R5/ 4.

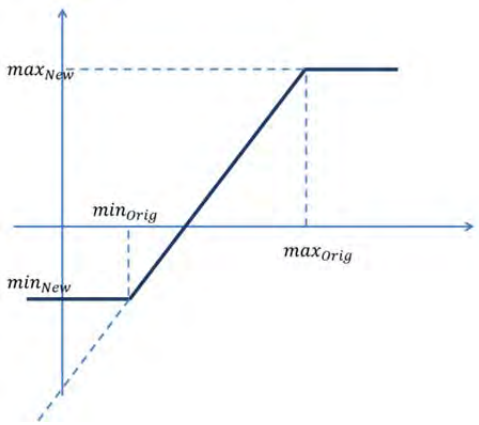


Figure R5/ 4 - Schematic of data transformation incurred with min max normalization.

We need the straight line equation: $y = m \cdot x + b$.

$$m = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}}$$

$$y = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot x + b$$

(max_{Orig}, max_{New}) ∈ straight line, so:

$$max_{New} = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot max_{Orig} + b$$

$$b = max_{New} - \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot max_{Orig}$$

$$b = \frac{max_{New} \cdot max_{Orig} - max_{New} \cdot min_{Orig} - max_{New} \cdot max_{Orig} + min_{New} \cdot max_{Orig}}{max_{Orig} - min_{Orig}}$$

$$b = -\frac{max_{New} \cdot min_{Orig} - min_{New} \cdot max_{Orig}}{max_{Orig} - min_{Orig}}$$

So,

$$y = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot x - \frac{max_{New} \cdot min_{Orig} - min_{New} \cdot max_{Orig}}{max_{Orig} - min_{Orig}}$$

$$y = \frac{1}{max_{Orig} - min_{Orig}} \left[(max_{New} - min_{New}) \cdot x - (max_{New} \cdot min_{Orig} - min_{New} \cdot min_{Orig} + min_{New} \cdot min_{Orig} - min_{New} \cdot max_{Orig}) \right]$$

$$y = \frac{1}{max_{Orig} - min_{Orig}} \left[(max_{New} - min_{New}) \cdot x - (max_{New} - min_{New}) \cdot min_{Orig} - min_{New} \cdot (min_{Orig} - max_{Orig}) \right]$$

$$y = \frac{x - min_{Orig}}{max_{Orig} - min_{Orig}} (max_{New} - min_{New}) + min_{New}$$

This is,

$$data_{normalized} = \frac{data - min_{Orig}}{max_{Orig} - min_{Orig}} (max_{New} - min_{New}) + min_{New}$$

62.2 4.2 Parzen window width

Chose the appropriate window size is not easy (Jenssen, Principe, Erdogmus, & Eltoft, 2006). However the Silverman's Rule gives us optimal window size estimation (Silverman, 1986, p. 87), for a multivariate normal case:

$$\sigma_{opt} = \hat{\sigma} \cdot h_{opt}$$

$$= \hat{\sigma} \cdot A(k) \cdot n^{-1/(d+4)} \quad \textbf{eq. R5/(47)}$$

$$= \hat{\sigma} \cdot \{4/(d+2)\}^{1/(d+4)} \cdot n^{-1/(d+4)}$$

Where $\hat{\sigma}^2$ is the average marginal variance:

$$\hat{\sigma}^2 = d^{-1} \sum_i s_{ii}$$

d is the dimension, in our case is the hidden vector dimension (L) and n is the size of train set (N).

62.3 4.3 Adaptive Learning Rate

An adaptive learning rate was implemented in all autoencoders. The rules used to define the adaptive learning evolution are the ones define in (Hagan, Demuth, & Beale, 1996, pp. 12-12). These rules are detailed here. This section describes the influence of the parameters *learning rate* (***lr***), *learning rate increase* (***lr_inc***), *learning rate decrease* (***lr_dec***) and *maximum performance increase* (***max_perf_inc***) (we use the same name adopted in MATLAB neural networks toolbox for these parameters). The changes to induce in the learning rate are assessed by the value of the ratio between the cost function in the current epoch and the cost function obtained in the previous epoch:

$$Ratio = \frac{F_t}{F_{t-1}} \qquad \textbf{eq. R5/(48)}$$

The changes induced into the autoencoder parameters, considering different values of this ratio and depending on the minimization / maximization objective function, may include **i)** changes in learning rate, **ii)** acceptance or rejection of the changes in weights and bias of the current epoch, **iii)** changes in the momentum term rate. So far, the model does not include changes on the momentum term rate (note that the momentum term is set to zero in all the results further presented).

Minimizing case

When minimizing the cost function, the adaptive learning algorithm evolves differently under three distinct scenarios. Table R5/ 1 summarizes the three scenarios.

Table R5/ 1 - Different cases of adaptive learning algorithm evolution when minimizing the cost function.

Ratio value	Assessment	Learning Rate	Weights and Bias
]-∞,1[	The cost function evolves as desired. The cost function is evolving in the desired direction (minimizing).	The learning coefficient is increased by *lr_inc* rate (in our case always 1.01 – augmenting 1%).	The changes performed in the current epoch are maintained (weights, bias)
[1,max_perf_inc]	The cost function evolves in the opposite direction.	The learning rate is not changed.	The changes of current epoch are maintained (weights

	"not severe" The cost function had evolved in the opposite direction to the one that was desired, but only 4%		and bias).
] max_perf_inc ,+∞[	The cost function evolves in the opposite direction. "too severe" The cost function evolved drastically in the opposite direction to the one expected (more than 4%)	The learning rate is cut by *lr_dec* (in our case always 0.5).	The changes performed in the current epoch are discarded (weights and bias).

The first scenario occurs when the Ratio value is lower than 1, meaning that the cost function decreased. In this case, the algorithm accepts the changes performed in the structure of the autoencoder (weights and bias upgrades), and the learning coefficient is increased by *lr_inc*. The second scenario ("not severe") occurs for Ratio values comprised between [1, *max_perf_inc*]. Besides the cost function evolution being opposite to the objective, the algorithm still accepts the changes made in this epoch and do not change the *lr*. The third scenario ("too severe") is considered for Ratio values higher than *max_perf_inc*. In this case the algorithm will discard the changes developed in the current epoch (this means that every change made under the current epoch is reverted, and the next epoch will have the same starting point). In this case the algorithm cuts the *lr* by *lr_dec*.

Maximizing case

The maximizing case (used in entropy of hidden layer and mutual information between inputs and hidden layer) is discussed in this section. Three possible situation may occur, as summarized in Table R5/ 2.

Table R5/ 2 - Different cases of adaptive learning algorithm evolution when maximizing the cost function.

CostFn / PrevCostFn	Assessment	Learning Rate	Weights and Bias
]-∞, 1/max_perf_inc [	The cost function evolves in the opposite direction. "too severe" The cost function evolved drastically in the opposite direction to the one expected (more than 1/ *max_perf_inc*).	The learning rate is cut by *lr_dec* (in our case always 0.5).	The changes performed in the current epoch are discarded (weights and bias).
[1/max_perf_inc, 1]	The cost function evolves in the opposite direction. "not severe" The cost function had evolved in the opposite direction to the one that was desired, but only till 1/ *max_perf_inc*.	The learning rate is not changed.	The changes of current epoch are maintained (weights and bias).
] 1,+∞[	The cost function evolves as desired. The cost function is evolving in the desired direction (maximizing).	The learning coefficient is increased by the *lr_inc* rate (in our case always 1.01 – augmenting 1%).	The changes performed in the current epoch are maintained (weights, bias)

The reasoning for these three situations is similar to the one previous described. The only difference is the use of the inverse value of *max_perf_inc* to define the limit between “not severe” and “severe” scenarios.

62.4 4.4 Stop Criteria

The stopping criterion is the number of epochs for all methods. The second part of the autoencoder includes an extra stop criterion: when the MSE of the train set is lower than a specified goal, the algorithm stops.

62.5 4.5 Neural Network Saturation

The saturation problem appears when the nonlinear activation functions reach its upper or lower saturation limits (El-Sharkawi, 1995). As El-Sharkawi explains (El-Sharkawi, 1995), any wide change in the input would produce no or minimal change in the output and the neurons in this case are paralyzed. El-Sharkawi also confirms that it is common and acceptable to have some neurons in the saturation region, but too many would render the neural network useless.

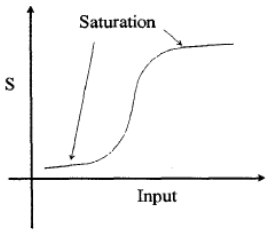


Figure R5/ 5 - Saturation regions of a Sigmoidal Function (from (El-Sharkawi, 1995)).

If network reaches saturation, the neurons must be randomly perturbed and the learning process continued. El-Sharkawi (El-Sharkawi, 1995) highlights the imperative need of including this saturation check in any NN training.

63 5 ITL methods pseudo-code

This section presents the pseudo-code for the four ITL methods developed. The code of each one of these methods is detailed in the further sections.

MaxE

```
Initialize[int numEpochs1; int numEpochs2; int DimIn; int DimHidden; int TestDim; int TrainDim; double momRate; actFcnFlatness; vec learnCoeff1; vec learnCoeff2; vec seed;]
Input [mat TrainDataSet]; Normalize [TrainDataSet]; Randomize [TrainDataSet]; Input [mat TestDataSet]; Normalize [TestDataSet]
Loop over learnCoeff1
[
    Loop over learnCoeff2
    [
        Loop over seed
        [
            Initialize [mat Weights1; vec Bias1;]; Initialize [mat HiddenTrain;];
            Loop over Epochs1
            [
                Loop over TrainDim (refer to instance yi)
                [
                    Loop over TrainDim (refer to instance yj)
                    [
                        Loop over DimHidden
                        [
                            Extract[vec yi; vec yj;]; Calculate [vec dij=yi-yj;];
                        ]
                        Calculus [Vij = G(yi-yj, 2sig^2);]; Initialize [mat delta_weights1;];
                        Loop over DimHidden
                        [
                            Calculus [dPI/dyi; dyi/(activation function);]
                            Loop over DimIn
                            [
                                Calculus [double momterm; dnet1/dweights1]; Calculus [mat delta_weights1;]
                                Update [Weights1;]
                            ]
                            Calculus [vec delta_Bias1;]; Update [Bias1;]
                        ]
                    ]
                ]
                initialize mat prev_HiddenTrain;
                prev_HiddenTrain = HiddenTrain; Update HiddenTrain;
                Verify [saturation conditions of first half];
                Calculus [entropy cost function;]
                Verify [Adaptive learnCoeff1 conditions]
            ]
            Initialize [mat Weigths2; vec Bias2;]; Initialize [mat OutputsTrain;]
            Loop over Epochs2
            [
                Loop over TrainDim (refer to instance yi)
                [
                    Loop over DimOut
                    [Calculus [dPI/dyi;]]
                    Loop over DimHidden
                    [
                        Loop over DimOut
                        [update Weigths2;]
                    ]
                    Loop over DimOut
                    [Update Bias2;]
                ]
                Loop over TrainDim
                [Update OutputsTrain;]
                Verify [saturation conditions of second half]; Calculate [mat ErrorMatrixTrain; MSE_train; MAE_train;]
                Verify [Stop criteria; Adaptive learnCoeff2 conditions]
                Calculate [mat HiddenTest; mat OutputsTest;]; Calculate [mat ErrorMatrixTest; MSE_test; MAE_test;]
            ]
            Print [Results;]
        ]
    ]
]
```

MaxMI_CS

Initialize[int numEpochs1; int numEpochs2; int DimIn; int DimHidden; int TestDim; int TrainDim; double momRate; actFcnFlatness; vec learnCoeff1; vec learnCoeff2; vec seed;]; Input [mat TrainDataSet]; Normalize [TrainDataSet]; Randomize [TrainDataSet]; Input [mat TestDataSet]; Normalize [TestDataSet]

Loop over learnCoeff1

[

| Loop over learnCoeff2

| [

| | Loop over seed

| | [

| | | Initialize [mat Weights1; vec Bias1;]; Initialize [mat HiddenTrain;]; Calculus [Vij_x; Vi_x; V_x;]; Initialize [CostFn; Prev_CostFn;];

| | | Loop over Epochs1

| | | [

| | | | Initialize mat Vij_y; vec dij_y; vec Vi_y; double aux_viy;

| | | | Loop over TrainDim (refer to instance yi)

| | | | [

| | | | | Loop over TrainDim (refer to instance yj)

| | | | | [

| | | | | | Loop over DimHidden [Calculate [vec dij_y=yi-yj;]]

| | | | | | Calculus [Vij_y = G(dij_y, 2sig^2);]

| | | | |]

| | | | | Calculus Vi_y;

| | | |]

| | | | Loop over TrainDim (refer to instance yi)

| | | | [

| | | | | Loop over TrainDim (refer to instance yj) [Calculus aux_vj; aux_vm;]

| | | |]

| | | | Loop over TrainDim (refer to instance yi)

| | | | [

```
| | | | | Loop over TrainDim (refer to instance yj)
| | | | | [
| | | | | | Loop over DimHidden
| | | | | | [
| | | | | | | extract y_i; y_j;
| | | | | | | Calculus dij; auxN1; auxN2; auxN3;
| | | | | | ]
| | | | | Calculus N1; N2; N3; D3;
| | | | | ]
| | | | | Loop over DimHidden
| | | | | [
| | | | | | Calculus Fi=dMI/dyi; dy/dw;
| | | | | | Loop over DimIn [Update Weighs1;]
| | | | | | Update Bias1;
| | | | | ]
| | | | ]
| | | | initialize mat prev_HiddenTrain;
| | | | prev_HiddenTrain = HiddenTrain;
| | | | Update HiddenTrain;
| | | | Verify [saturation conditions of first half];
| | | | Calculus [entropy cost function;]
| | | | Verify [Adaptive learnCoeff1 conditions]
| | | ]
| | | Initialize [mat Weigths2; vec Bias2;]
| | | Initialize [mat OutputsTrain;]
| | | Loop over Epochs2
```

| | | [

| | | | Loop over TrainDim (refer to instance yi)

| | | | [

| | | | | Loop over DimOut [Calculus [dPI/dyi;]]

| | | | | Loop over DimHidden

| | | | | [

| | | | | | Loop over DimOut [update Weigths2;]

| | | | |]

| | | | | Loop over DimOut [Update Bias2;]

| | | |]

| | | | Loop over TrainDim [Update OutputsTrain;]

| | | | Verify [saturation conditions of second half]; Calculate [mat ErrorMatrixTrain; MSE_train; MAE_train;]

| | | | Verify [Stop criteria; Adaptive learnCoeff2 conditions]

| | | | Calculate [mat HiddenTest; mat OutputsTest;]; Calculate [mat ErrorMatrixTest; MSE_test; MAE_test;]

| | |]

| | | Print [Results;]

| |]

|]

]

MaxMI_ED

Initialize[int numEpochs1; int numEpochs2; int DimIn; int DimHidden; int TestDim; int TrainDim; double momRate; actFcnFlatness; vec learnCoeff1; vec learnCoeff2; vec seed;]

Input [mat TrainDataSet]

Normalize [TrainDataSet]

Randomize [TrainDataSet]

Input [mat TestDataSet]

Normalize [TestDataSet]

Loop over learnCoeff1

[

| Loop over learnCoeff2

| [

| | Loop over seed

| | [

| | | Initialize [mat Weights1; vec Bias1;]

| | | Initialize [mat HiddenTrain;]

| | | Calculus [V; Vj; Vij; Vi; Cij;]

| | | Initialize [CostFn; Prev_CostFn;]

| | | Loop over Epochs1

| | | [

| | | | Initialize vec y_i; vec y_j; vec dij2; double Vij_hid; mat delta_weights1;

| | | | Loop over TrainDim (refer to instance yi)

| | | | [

| | | | | Loop over TrainDim (refer to instance yj)

| | | | | [

| | | | | | Loop over DimHidden

| | | | | | [

| | | | | | | Calculate [vec y_i; vec y_j; vec dij2=yi-yj;]

| | | | | |]

| | | | | | Calculus [Vij_hid = G(dij2, 2sig^2);]

| | | | | | Loop over DimHidden

```
| | | | | | [
| | | | | | | calculus derv_i; derv_j; dV/dyi; dV/dyj; dy/dw;
| | | | | | | Loop over DimIn [update Weigths1;]
| | | | | | | Update Bias1;
| | | | | | ]
| | | | | ]
| | | | ]
| | | | initialize mat prev_HiddenTrain;
| | | | prev_HiddenTrain = HiddenTrain;
| | | | Update HiddenTrain;
| | | | Verify [saturation conditions of first half];
| | | | Calculus [entropy cost function;]
| | | | Verify [Adaptive learnCoeff1 conditions]
| | | ]
| | | Initialize [mat Weigths2; vec Bias2;]
| | | Initialize [mat OutputsTrain;]
| | | Loop over Epochs2
| | | [
| | | | Loop over TrainDim (refer to instance yi)
| | | | [
| | | | | Loop over DimOut    [Calculus [dPI/dyi;]]
| | | | | Loop over DimHidden
| | | | | [
| | | | | | Loop over DimOut [update Weigths2;]
| | | | | ]
| | | | | Loop over DimOut [Update Bias2;]
```

| | | |]

| | | | Loop over TrainDim [Update OutputsTrain;]

| | | | Verify [saturation conditions of second half];

| | | | Calculate [mat ErrorMatrixTrain; MSE_train; MAE_train;]

| | | | Verify [Stop criteria; Adaptive learnCoeff2 conditions]

| | | | Calculate [mat HiddenTest; mat OutputsTest;]

| | | | Calculate [mat ErrorMatrixTest; MSE_test; MAE_test;]

| | |]

| | | Print [Results;]

| |]

|]

]

Uni

Initialize[int numEpochs1; int numEpochs2; int DimIn; int DimHidden; int TestDim; int TrainDim; double momRate; actFcnFlatness; vec learnCoeff1; vec learnCoeff2; vec seed;]

Input [mat TrainDataSet]

Normalize [TrainDataSet]

Randomize [TrainDataSet]

Input [mat TestDataSet]

Normalize [TestDataSet]

Loop over learnCoeff1

[

| Loop over learnCoeff2

| [

| | Loop over seed

| | [

| | | Initialize [mat Weights1; vec Bias1;]

| | | Initialize [mat HiddenTrain;]

| | | Initialize [CostFn; Prev_CostFn;]

| | | Loop over Epochs1

| | | [

| | | | Initialize [mat delta_weights1;]

| | | | Loop over DimHidden (refers to the first neuron of hidden layer)

| | | | [

| | | | | Loop over DimHidde (refers to the second neuron within hidden layer)

| | | | | [

| | | | | | Initialized [mat Vij_x; mat Vij_y; vec Vj_x; vec Vj_y;]

| | | | | | Loop over TrainDim (refer to instance yi)

| | | | | | [

| | | | | | | Loop over TrainDim (refer to instance yj) [Calculus [dij_x; dij_y; Vij_x; Vij_y;]]

| | | | | | | Calculus [Vj_x; Vj_y;]

| | | | | |]

| | | | | | Calculus [V_x; V_y;]

| | | | | | Calculus [D1; D2_x; D2_y;]

| | | | | | Loop over TrainDim (refer to instance yi)

| | | | | | [

| | | | | | | Loop over TrainDim (refer to instance yj)

| | | | | | | [

| | | | | | | | Calculus [x_i; x_j; y_i; y_j; dij_x; dij_y;]

| | | | | | | | Calculus [N1_x; N2_x; N3_x; N1_y; N2_y; N3_y; D3;]

| | | | | | |]

| | | | | | | Calculus [Fi_x; Fi_y;]

| | | | | | | Calculus [dxj/activation function; dyj/activation function;]

| | | | | | | Loop over DimIn [update Weigths1;]

| | | | | | | Update Bias1;

| | | | | |]

| | | | |]

| | | |]

| | | | initialize mat prev_HiddenTrain;

| | | | prev_HiddenTrain = HiddenTrain;

| | | | Update HiddenTrain;

| | | | Verify [saturation conditions of first half];

| | | | Calculus [entropy cost function;]

| | | | Verify [Adaptive learnCoeff1 conditions]

| | |]

| | | Initialize [mat Weigths2; vec Bias2;]

| | | Initialize [mat OutputsTrain;]

| | | Loop over Epochs2

| | | [

| | | | Loop over TrainDim (refer to instance yi)

| | | | [

| | | | | Loop over DimOut [Calculus [dPI/dyi;]]

| | | | | Loop over DimHidden

| | | | | [

| | | | | | Loop over DimOut [update Weigths2;]

| | | | |]

| | | | | Loop over DimOut [Update Bias2;]

| | | |]

| | | | Loop over TrainDim [Update OutputsTrain;]

| | | | Verify [saturation conditions of second half];

| | | | Calculate [mat ErrorMatrixTrain; MSE_train; MAE_train;]

| | | | Verify [Stop criteria; Adaptive learnCoeff2 conditions]

| | | | Calculate [mat HiddenTest; mat OutputsTest;]

| | | | Calculate [mat ErrorMatrixTest; MSE_test; MAE_test;]

| | |]

| | | Print [Results;]

| |]

|]

]

64 6 ITL methods code documentation

This section details the code developed to train autoencoders using two separated parts. The first part is trained with an ITL criteria, implemented using a gradient approach. The second part evolves following the classic propagation gradient criteria. These autoencoders share the same general structure, and only differ concerning the specific criteria depending on the ITL method applied. Therefore, in this section the shared structure among all ITL autoencoders is described. The specific criterion used in each method is detailed in further sections. All autoencoders are implemented using the **batch mode** only.

Auxiliary Functions

actFcn

This function implements the choice for the activation function, with **activation** and **actFcnFlatness** as input parameters, and this function returns scalar variable. We can choose between 3 functions: Logistic, Hyperbolic Tangent and Linear.

$$Logistic = \frac{1}{1 + \mathrm{e}^{-\frac{\mathbf{activation}}{\mathbf{actFcnFlatness}}}}$$

$$Hyperbolic\ Tangent = \frac{2}{1 + e^{-\frac{\mathbf{activation}}{\mathbf{actFcnFlatness}}}} - 1$$

At MATLAB, $Hyperbolic\ Tangent = \frac{2}{1+e^{-2*\mathbf{activation}}} - 1$, considering $\mathbf{actFcnFlatness} = {}^{1}/_{2}$.

$$Linear = \mathbf{activation}$$

The parameter values adopted were the same as the ones pre-specified in MATLAB software.

DervActFcn

This function implements the derivative for the chosen activation function, with **output** and **actFcnFlatness** as input parameters.

$$Derivative\ Logistic = \frac{(\mathbf{output} * (1 - \mathbf{output}))}{\mathbf{actFcnFlatness}}$$

$$Derivative\ Tangent = \frac{(1 - \mathbf{output^2})}{\mathbf{2} * \mathbf{actFcnFlatness}}$$

At MATLAB, $Derivative\ Tangent = (1 - \mathbf{output^2})$, considering $\mathbf{actFcnFlatness} = {}^{1}/_{2}$.

$$Derivative\ Linear = 1$$

To our runs, $Derivative\ Hyperbolic\ Tangent$ for MATLAB is chosen according to the choice made for the activation function. The result of this function is a variable of type double, **tmpDerivative**.

PCA

This function implements the intrinsic function **princomp** of Library Armadillo for C++. The input parameters are **TrainSet_PCA** and *DimIn*. The output of *Armadillo* function **princomp** is a matrix (*DimInx DimIn*) of coefficients of principal components (*DimIn*). The result of this function is a variable of type matrix, **Weights_PCA**.

```
//Auxiliar Functions

double  actFcn(double activation, double actFcnFlatness)

  [

    double output = ( 2.0 / ( 1.0 + exp( - 2.0* activation ) ) ) - 1.0; //matlab

    return output;

  ]
```

```
double  DervActFcn(double output, double actFcnFlatness)
   [
      double tmpDerivative;
      tmpDerivative = ( 1.0 - pow( output , 2.0 ) );
      return tmpDerivative;
   ]
mat PCA(mat TrainSet_PCA, unsigned int DimIn)
[
   mat Weights_PCA(DimIn,DimIn);
   Weights_PCA = princomp(TrainSet_PCA);
   return Weights_PCA;
]
```

Auxiliary parameters

The **goal** parameter defines the boundary from where the autoencoder should stop. Thus, if the MSE obtained with the autoencoder achieves a value lower than the parameter goal, the algorithm stops.

```
double lr_dec = 0.5;
double lr_inc = 1.05;
double max_perf_inc = 1.04; //performance/racio (=CostFunction/Prev_CostFunction) a partir do qual nao se actualizam pesos
double goal = 0.00000001; //como no matlab
```

The other three parameters (*lr_dec*, *lr_inc* and *max_perf_inc*) are also specified, in this example the values applied by default in MATLAB were adopted.

Vector learnCoeff

The code is intended to run several simulations, each one with a different learning coefficient. This is achieved with the definition of a **vector "learnCoeff"**, which includes all the desired values to test the learning coefficient. The construction of this vector can be done with a loop (when the values differ on the same slope) or with the individual inputs (when the values to test are distinct).

```
//###defining a vector containing the learning coefficients to test the reducer###
vector <double> learnCoeff1;
learnCoeff1.push_back(0.005);
learnCoeff1.push_back(0.05);
learnCoeff1.push_back(0.5);
learnCoeff1.push_back(5);

vector <double> learnCoeff2;
learnCoeff2.push_back(0.005);
learnCoeff2.push_back(0.05);
learnCoeff2.push_back(0.5);
learnCoeff2.push_back(5);

unsigned int DimLC1;
DimLC1 = learnCoeff1.size();

unsigned int DimLC2;
DimLC2 = learnCoeff2.size();
//###end of defining a vector containing the learning coefficients to test the reducer###
```

The **value "DimLC1"** measures the dimension of the vector containing the learning coefficients to test first part. This variable is further applied to define one of the main loops. The **value "DimLC2"** measures the dimension of the vector containing the learning coefficients to test second part.

Main Results File

The code creates a *csv file*, with a specific name for each method, to further include the main information of each autoencoder tested. This file will include information on the **i)** starting values of the two learning coefficients used (for the first part and second part), **ii)** the final values of these coefficients (they evolve following an adaptive algorithm, which decreases their value whenever the respective cost function evolves

to the opposite direction to the optimization objective), **iii)** the boost random seed used, **iv)** the final information potential obtained in the end of the first part (e.g. the final value of entropy), **v)** the mean absolute error MAE and **vi)** the mean squared error MSE between outputs and inputs of the autoencoder.

```
ofstream Resultados_LrCoeff;//ficheiros de saida de dados

vec MaxMI(DimLC);

string name ("Results_MaxMI_CS.csv");

Resultados_LrCoeff.open(name);
```

Fixed Parameters

The main parameters are defined together. The variables **numEpochs1** and **numEpochs2** define the number of epochs that the code will execute in each part of the autoencoder (first part relates to the ITL criteria, and the second part relates to the classic propagation).

DimIn is the number of neurons at the entrance of the autoencoder. **DimHidden** is the number of neurons composing the hidden layer. **DimOut** relates to the number of neurons composing the third layer (when construction autoencoders, DimOut is equal to DimIn). These three variables are "integers".

The variables **dDimIn, dDimHidden** and **dDimOut** are "doubles", with similar meaning as the correspondent "integers" (the joint use of integer and double is not a good idea for some functions - therefore we never mix integers with doubles).

The number of instances stored to integrate the test data set is defined with the "integer" **TestDim,** and **dTestDim** is the "double" corresponding to the "integer" TestDim. The number of instances integrating the Train data set is defined within the "integer" **TreinoDim,** being the variable **dTreinoDim** its corresponding "double".

The code includes the implementation of momentum rate (Miranda, 2007), only for the second half of the autoencoder, when the classic propagation is applied. The momentum rate (α) is specified in the variable **momRate**.

The variable **actFcnFlatness** refers to the activation function flatness (Haykin, 1999). In this work this variable is always defined with the value of 0.5 as this is the default value defined in the software MATLAB for this variable.

The variables **SatLimit** and **ThresholdSat** are used in the Saturation procedure. It is possible to choose two different initializations of the second part weights: a random initialization or the transpose matrix of the

final weights obtained in the first part. This choice is made by defining (uses the transpose matrix) or commenting (uses the random initialization) the **InicPesosTranspose** variable. Next, **Normalization** variable allows to choose between (i) when defined, the code normalizes both data sets (train data set and test data set) by entry; (ii) when undefined, the code do not perform any normalization procedure. The variable **WeightsInitialization_PCA** should be defined when the weights initialization (1st half) intended to be with Principal Component Analysis (PCA). When not defined the weights are initialized randomly with uniform distribution.

```
// #### Main Parameters Definition #####

unsigned int numEpochs1 = 2000;

unsigned int numEpochs2 = 2000;

unsigned int DimIn = 24; // instances Dimension = first layer Dimension

double dDimIn = 24.0;

unsigned int DimHidden = 12; // output Dimension = hidden layer Dimension

double dDimHidden = 12.0;

unsigned int DimOut = DimIn; // instances Dimension = output layer Dimension

double dDimOut = dDimIn;

unsigned int TestDim = 500;

double dTestDim = 500.0;

unsigned int TrainDim = 1000; // Numbers of instances/examples in train set

double dTrainDim = 1000.0;

double momRate = 0.0; //momentum coef

double actFcnFlatness = 0.5; // For Activation Function

double SatLimit=0.9; //o valor da função de activação a partir do qual se considera saturado

double ThresholdSat=0.3; //a percentagem máxima permitida de valores saturados a partir da qual se faz uma perturbação aleatória aos pesos


// #### Define Inicialização dos pesos da 2ªmetade
```

```
#define InitWeightsTranspose // use Transpose Matrix

// if this is undefined, we have Random Matrix


// #### Define normalization of the data sets

#define Normalization // if this is undefined do nothing;

// if this is defined, do the normalization by entrance


// #### Define Pesos Iniciais

#define WeightsInitialization_PCA //do PCA Initialization of Weights1;

// if this is undefined, do the Random Weights1 Initialization

// #### End of Main Parameters Definition #####
```

Random numbers generators

The code uses the boost libraries (see www.**boost**.org) to define two functions that generate random numbers. The function **aux_aleat(rng)** generates uniform real numbers between the interval [0,1], the function **aleat(rng)** generates uniform realm numbers between the interval [-1,1].

```
//#### random functions definition ####

boost::random::mt19937 rng;

boost::random::uniform_real_distribution<> aux_aleat(0,1);

boost::random::uniform_real_distribution<> aleat(-1,1);

//#### END of random functions definition ####
```

Randomizing the instances in train data set

The train data is a set of instances, or particles (each instance has the same size as the first layer of the autoencoder). The order in which the particles are covered is defined randomly (but the same order is used in every comparison of different methods, as the random seeds considered are the same). To create this random instance selection, the vector **indices** is created, which contains different integer numbers between 0 and *TreinoDim*, and is further used to address the instances position on the train set.

```
// ##### Randomize TrainSet ######
```

```
rng.seed(3578);

vec RandPosition(TrainDim);

for (unsigned int i=0; i<TrainDim;i++)

[RandPosition(i) = aux_aleat(rng);]

uvec indices = sort_index(RandPosition);

// ##### END Randomize TrainSet ######
```

Reading data from a csv file

The code reads two *csv files*: the train data set and the test data set. The code used to read both files is similar. This way, a provisional matrix is created (**ordered_TrainSet**) which contains all train instances organized in the same order as the input file. A new matrix is created, the **TrainSet** (the one going to be applied in the further calculations). This matrix reads the information of the *ordered_TrainSet* matrix using the *indices* vector to organize the instances.

```
//##### Reading Training Data from csv file #######

ifstream indata;

double f;

string s="";

mat TrainSet(TreinoDim,DimIn); // train set matrix

mat ordered_TrainSet(TreinoDim,DimIn); // train set matrix

indata.open("NovoTipoAC_1.csv");

if(!indata) [ cerr << "Error: file could not be opened" << endl; exit(1);]

vector < double > auxiliar;

while(getline(indata, s, ';'))

[

   stringstream fs( s );

   fs >> f;

   auxiliar.push_back(f);
```

```
]

for(unsigned int k=0; k<TreinoDim; k++)
[
   for (unsigned int j=0; j<DimIn; j++)
   [ordered_TrainSet(k,j) = auxiliar[(k*DimIn+j)];]
]
indata.close();
(DATA NORMALIZATION BLOCK – See next Section)

for(unsigned int k=0; k<TreinoDim; k++)
[
   for (unsigned int j=0; j<DimIn; j++)
   [TrainSet(k,j) = norm_ordered_TrainSet(indices(k),j);]
]
ordered_TrainSet.clear();
norm_ordered_TrainSet.clear();
//##### END Reading Training Data from csv file #######
```

Analogously, the matrix **TestSet** is created, which contains a set of instances to assess the best number of epochs to train the autoencoder. There is no need to randomize the instances of the test data set, since they will not influence the train, and the error measured is not changeable with their order (this matrix is used only after the training is complete).

Data Normalization

When the variable Normalization is defined, the code will perform the normalization by entry. Since the new range is $[-1, 1]$ (according to the activation function), the first step is obtain the original range by

entrance. In that way, **min_train** is the vector that contains the minimum of original data (*ordered_TrainSet*) by column and **max_train** is the vector that contains the maximum by column.

ordered_TrainSet = train data set from csv file (dim = *TrainDim x DimIn*)

min_train = minimum value by columns of data matrix *ordered_TrainSet* (dim = 1 x *DimIn*)

max_train = maximum value by columns of data matrix *ordered_TrainSet* (dim = 1 x *DimIn*)

The normalization by $\mathrm{column}(j)$ is done using the corresponding minimum and maximum value, this is, using **min_train(j)** and **max_train(j)**.

```
// ##### Data normalization #####
#ifdef Normalization //performs the normalization by entrance
rowvec max_train = max(ordered_TrainSet,0);
rowvec min_train = min(ordered_TrainSet,0);
mat norm_ordered_TrainSet(TrainDim,DimIn);
for(unsigned int tr=0; tr<TrainDim; tr++)
[
for (unsigned int in=0; in<DimIn; in++)
   [norm_ordered_TrainSet(tr,in)=(2*ordered_TrainSet(tr,in)-max_train(in)-min_train(in))/(max_train(in)-
min_train(in));]
]
#endif
//##### End Data normalization #####
```

Similar code and description for test data set, **aux_TestSet**, normalization with correspondent minimum **min_test** and maximum **max_test**.

Learning Coefficient and seed loops

There are **three main loops**: for each combination of **i)** learning coefficient 1, **ii)** learning coefficient 2 and **iii)** seed, an autoencoder is created. Since the learning coefficients were specified with vectors, the two

loops concerning learning coefficients relate to the position of *learnCoeff* vector to use. In the further example the same *learnCoeff* vector is used for both NN parts, meaning that all combinations of the learning coefficients included in that vector are to be tested. Nevertheless different combinations may be tested, using distinct vectors for the first and second parts.

```
for (unsigned int v=0; v<DimLC1; v++)
[
   for (unsigned int p=0; p<DimLC2; p++)
   [
Detailed_Output file
      for(unsigned int seed=800; seed<850; seed+=5)
      [
         // #### Main Parameters Definition #####
         double lrCoeff1 = learnCoeff1[v];
         double lrCoeff2 = learnCoeff2[p];
         // #### End of Main Parameters Definition #####
Next sub-sections: i) Weights initialization, Bias initialization, etc.
      ] //end of seed loop
   ] //end of learning coefficient 2 loop
] //end of learning coefficient 1 loop
```

The next sub-sections describe what is included within these three main loops.

Weights (1st Half) initialization

The weights of the first part of the autoencoder can be initialized with PCA or random numbers. These weights are created within a matrix, the **Weights1** (for first part) and **Weights2** (for second part) matrices (dimension of *DimIn* x *DimOut*, *DimOut* x *DimIn*, respectively). Matrices **prev_Weights1** and **prev_Weights2** (dimension of *DimIn* x *DimOut*, *DimOut* x *DimIn*, respectively) of weights are defined to store in each epoch the weights of the previous epoch, for first and second part, respectively.

PCA

PCA initialization is stored before loops related with learning coefficient and seed. Matrix **Coeff_PCA** store the coefficients of the principal components of train set, **TrainSet**, using PCA function defined before, that performs the Principal Components Analysis. In that method, **Weights1** stores, in its initialization, the coefficients of first *DimHidden* principal components. **Weights2** is defined as the transpose of **Weights1**.

```
// PCA
#ifdef WeightsInitialization_PCA
mat Coeff_PCA(DimIn,DimIn);
mat Weights1(DimIn, DimHidden);
mat Weights2(DimHidden, DimIn);
Coeff_PCA = PCA(TrainSet, DimIn);
Weights1 = Coeff_PCA.submat(span(), span(0, DimHidden-1));
Weights2 = trans(Weights1);
#endif
// END PCA
```

Random

The random numbers are repeatable for every experiment, as the seed used is defined in the third main loop (and this information is stored in the general results file as well).

```
mat prev_Weights1(DimIn,DimHidden); prev_Weights1.zeros();
//##### Weights1 Inicialization #####
mat Weights1(DimIn,DimHidden);
rng.seed(seed);
for (unsigned int in=0; in<DimIn; in++)
[
   for (unsigned int hid=0; hid<DimHidden; hid++)
   [Weights1(in,hid) = aleat(rng);]
]
```

//##### End of Weights1 Inicialization #####

Bias (1st Half) initialization

The bias of the first half of the NN are always initialized with random numbers, using the *aleat(rng)* function. The vector **Bias1** (dimension *DimHidden*) stores the bias values for each epoch, the vector **prevBias1** (dimension *DimHidden*) stores the Bias values of the previous epoch (initialized with zeros).

//##### Bias1 Inicialization #####

vec Bias1(DimHidden);

vec prevBias1(DimHidden); prevBias1.zeros();

for (unsigned int hid=0; hid<DimHidden; hid++)

[Bias1(hid) = aleat(rng);]

//##### End of Bias1 Inicialization #####

Calculating Hidden matrix

Once the matrix *Weights1* and the vector *Bias1* are initialized, the model estimates the respective hidden instances (for the first part of the NN). This way, all train instances contained in *TrainSet* are transformed and stored in the matrix **HiddenTrain** (dimension *TrainDim* x *DimHidden*). For each exit neuron, an induced local field is calculated with an auxiliary variable *aux1*, according to eq. R5/(49).

$$v_k = \sum_{i=1}^{DimIn} w_{ki} x_i + b_k$$

Or

$$net_k = \sum_{i=1}^{DimIn} w_{ki} x_i + b_k$$

eq. R5/(49)

In expression eq. R5/(49), the induced local field v_k for an exit neuron k is defined as the sum of all outputs from the previous layer (x_i) multiplied with the respective weights linked to neuron k (w_{ki}), and the bias weight associated with the neuron k (b_k). For further theoretical information, see (Haykin, 1999, pp. 34-36). This induced local field is then applied to the activation function previously defined, using the function **actFcn** (v_k, actFcnFlatness).

//##### Hidden Matrix == FeedForward Step #####

```
mat HiddenTrain(TrainDim,DimHidden);  //cria matriz de HiddenTrain de (DimInensao padrões X tamanhoOutput)
double aux1 = 0.0;
for (unsigned int tr=0; tr<TrainDim; tr++)
[
   for(unsigned int hid=0; hid<DimHidden;hid++)
   [
      for(unsigned int in=0; in<DimIn;in++)
      [aux1 += ((TrainSet(tr,in))*Weights1(in,hid));]
      HiddenTrain(tr,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness); aux1 = 0.0;
   ]
]
//##### End of Calculating Hidden Matrix #####
```

Estimating the sigma value

The ITL methods used in these autoencoders need an estimation of the sigma value to be adopted. In this part of the model, the variable **sig2** is calculated, which is the estimation of σ^2, to be used in the further calculations. This sigma is calculated with base on the first *HiddenTrain* matrix, and is used in all epochs of the first part. This estimation follows the Silverman's Rule for the multivariate case (Silverman, 1986).

```
//##### Sigma Calculation #####
// Sigma Calculation By Silverman's Rule
// Silverman, 1998 (First Edition 1986), page87
vec meane (DimHidden); vec meane2 (DimHidden);
meane.zeros(); meane2.zeros();
for (unsigned int tr=0; tr<TrainDim; tr++)
[
   for (unsigned int hid=0; hid<DimHidden; hid++)
```

```
    [
        meane(hid) += HiddenTrain(tr,hid)/dTrainDim;
        meane2(hid) += HiddenTrain(tr,hid)*HiddenTrain(tr,hid);
    ]
]
double aux2 = 0.0;
for (unsigned int hid=0; hid<DimHidden; hid++)
[aux2 += meane2(hid)/(dTrainDim-1.0) - meane(hid)*meane(hid)* dTrainDim/(dTrainDim-1.0);//sum of marginals variance]
double sdev = aux2 / (dDimHidden);
double Ak = pow((4.0/(dDimHidden+2.0)), 1.0/(dDimHidden+4.0));
double h_opt = Ak * pow(static_cast<double> (dTrainDim), -1.0/(dDimHidden+4.0));
double sig = sqrt(sdev) * h_opt;
double sig2 = sig*sig;
//##### End of Sigma Calculation #####
```

Maximum Learning Coefficient Calculation

The learning coefficient should respect the threshold defined in eq. R5/(50), where η_1 is the learning coefficient for the first part of the NN, and λ_{MAX} is the maximum eigenvalue obtained from the variance and covariance matrix of the train data (Hagan et al., 1996).

$$\eta_1 \leq \frac{1}{2 \times \lambda_{MAX}} \qquad \textbf{eq. R5/(50)}$$

This threshold is calculated in the model and stored in the variable **Max_lrCoeff1**. A detailed overview on this topic is provided in Hagan et al. (1996, pp. 9-6).

```
//##### Maximum Learning Coefficient Calculation #####
// CRITERIUM: lrCoeff < 1/ 2*lambdaMax
// cov = 1/(n-1) sum( (xi-xm)(xj-xm) )
```

```
mat C1(DimIn,DimIn); //, C2(DimIn,DimIn);

C1 = cov(TrainSet,TrainSet);

vec eigval1 = eig_sym(C1);

double lambda1 = max(eigval1);

double Max_lrCoeff1 = 1.0 / ( 2.0*lambda1 );

//##### End of Maximum Learning Coeficient Calculation  #####
```

Cost functions variables

Two variables are initialized before the epochs 1 loop: the **Prev_CostFn** variable (initialized with zero) that stores the cost function value of the previous epoch, and the **CostFn** variable (initialized with a big number) that stores the cost function calculated in each epoch. These two variables are needed outside the epochs 1 loop, as they are further used to assess the stop criteria and to define the adaptive evolution of the learning coefficient 1.

```
double Prev_CostFn = 0.0;

double CostFn = 100000.0;
```

Epochs loop (first part)

The epochs loop is declared as follows. The next sub-sections detail the processes that are included within this loop.

```
for (unsigned int epoch=0; epoch<numEpochs1; epoch++)

[//(…)]
```

Declaring variables

Declaration of the auxiliary vectors **y_i**, and **y_j** that will store the instances under assessment over the next loops, and the auxiliary vector **dij** that will store the divergence between the instances under assessment.

```
//##### BackForward Step == MaxE Criterium #####

vec y_i(DimHidden), y_j(DimHidden), dij(DimHidden);

double Vij_out;
```

Traveling over the instances

So far, all the code presented was shaded with grey, as it is common for all ITL methods. At this point, we will introduce "Back Forward Step" block code shaded with four colors, each one referring to an ITL model. The theoretical detail of these processes can be consulted at section 3 of this report, for each ITL method.

Table R5/ 3 – Color code adopted for each ITL method.

	MaxE
	MaxMI_CS
	MaxMI_ED
	MinMI_CS_Uni

MaxE sub-code

Declaration of the auxiliary vectors **y_i**, and **y_j** that will store the instances under assessment over the next loops, and the auxiliary vector **dij** that will store the divergence between the instances under assessment. Declaration of the **Vij_out** constant that will store the value of the Gaussian of the output neurons in hidden layer, $G\left(y_i - y_j, 2\sigma^2 I\right)$. Also, the declaration of the matrix **aux_deltas** that will store the value of the total variation in weights into that moment.

The next two loops (unsigned int **i** and **j** variables) cross the instances by pairs. In the third loop (unsigned int **hid**), (for each output neuron, this is each component of vectors), the calculation of **y_i** (instance i), **y_j** (instance j) and **dij** vectors are concretized.

Vij_out calculation is made based on *Vij_function* $(= G\left(y_i - y_j, 2\sigma^2 I\right))$. Declaration of matrix **delta_weights1** that will store the weights' modifications (at first part of autoencoder). The loop over *DimHidden* (unsigned int **hid**) calculates new weights and bias for each output neuron. For that, derivatives of activation function for instance **i** are stored at variable **derv_i**, using the *DervActFcn* function. From expressions eq. R5/(11) and eq. R5/(12):

$$\frac{\partial}{\partial \omega} V(Y) = \sum_{i=1}^{N} [F_i]^T \frac{\partial y_i}{\partial \omega}$$

$$[F_i]^T \frac{\partial y_i}{\partial \omega} = \left(-\frac{1}{N^2 \sigma^2} \sum_{j=1}^{N} V_{ij} d_{ij} \right) \cdot \left(\frac{\partial y_i}{\partial net_i} \frac{\partial net_i}{\partial \omega} \right)$$

So, in analogy with theory below,

derv_i$= \frac{\partial y_i}{\partial net_i}$

$$\mathbf{error\_i} = F_i \cdot \frac{\partial y_i}{\partial net_i} = \left(-\frac{1}{N^2 \sigma^2} \sum_{j=1}^{N} V_{ij} d_{ij} \right) \cdot \mathbf{derv\_i}$$

The loop over *DimIn* (unsigned int **in**) identifies the weight associated to each combination of hidden neuron (unsigned int **hid**) and input neuron (new weights and bias **in**). For each weight identified in this loop, momentum term is calculated (despite **momRate** parameter defined equal to zero). Then **delta_weights1** is calculated according to expression eq. R5/(3):

$$\mathbf{delta\_weights1} = \Delta\omega_{in,hid} = \eta \frac{\partial V}{\partial \omega_{in,hid}} + momterm$$

From the former equations, the ratio $\frac{\partial net_i}{\partial \omega_{in,hid}}$ is the only component to be calculated. Considering the definition of net_i in expression eq. R5/(49), follows that:

$\frac{\partial net_i}{\partial \omega_{in,hid}} = x_{in} = \mathbf{TrainSet(i, in)}$.

So,

$$\mathbf{delta\_weights1} = \eta \frac{\partial V}{\partial \omega_{in,hid}} + momterm$$

$$= \eta \sum_{i=1}^{N} [F_i]^T \frac{\partial y_i}{\partial \omega_{in,hid}} + momterm$$

$$\mathbf{delta\_weights1(in, hid)} = \eta \cdot F_i \cdot \left(\frac{\partial y_i}{\partial net_i} \frac{\partial net_i}{\partial \omega_{in,hid}} \right) + momterm$$

$$= \eta \cdot \mathbf{error\_i} \cdot \frac{\partial net_i}{\partial \omega_{in,hid}} + momterm$$

$$= \eta \cdot \mathbf{error\_i} \cdot \mathbf{TrainSet(i, in)} + momterm$$

In same loop, for each weight, **aux_deltas** (a control variable) is updated as the sum of latest **delta_weights1**; **prev_Weights1** is set equal to the actual weights (from the last epoch) and **Weights1** is a matrix of new weights that sums to the latest weights their variations (**delta_weights1**).

Outside of this loop, i.e., only considering the loop in **hid** (hidden neurons), identical process is done for update bias value. The variable **prevBias1** stores the last bias (previous epoch) and **Bias1** stores the bias values corresponding to the actual epoch. The values of Bias1 are updated by adding to previous bias values the bias variation. This reasoning is similar for both halves of the autoencoder.

$$\mathbf{Bias1}(\text{hid}) = \mathbf{prev\_Bias1}(\text{hid}) + \Delta b_{hid}$$

(At code, this equation is equivalent to $\mathbf{Bias1} \mathrel{+}= \Delta b_{hid}$)

$$\Delta b_{hid} = \eta \frac{\partial V}{\partial b_j} + momterm = \eta \cdot \mathbf{error\_i} + momterm$$

To ensure that the model minimizes the information potential, the negative sign (-1) is multiplied to the calculation of **delta_weights1** and **Bias1**.

```
for (unsigned int i=0; i<TrainDim; i++) //loop over instance i (Cross samples)
[
    for (unsigned int j=0; j<TrainDim; j++) //loop over instance j (Cross samples)
    [
        for(unsigned int hid=0; hid<DimHidden; hid++) //for each neuron within hidden layer
        [
            y_i(hid) = HiddenTrain(i,hid); //instance i extraction
            y_j(hid) = HiddenTrain(j,hid); //instance j extraction
            dij(hid) = y_i(hid) - y_j(hid); // the difference bettween instances i and j
        ]
        Vij_out = Vij_function(dij, DimHidden, sig2); // Vij = G(yi-yj, 2sig^2)

        // dV/Dw = dV/dyi * dyi/dw
        mat delta_weights1(DimIn,DimHidden);
        for(unsigned int hid=0; hid<DimHidden; hid++) //for each output neuron: derivative
        [
            // Act Function Derivative @ Y_y
            double derv_i = DervActFcn(y_i(hid), actFcnFlatness); //novo nome
            //dV/dyi = dij[hid] * Vij_out * (-1.0/(2.0*sig2*TrainDimIn*TrainDimIn))
            double error_i = dij(hid) * Vij_out * (-1.0/(2.0*sig2*dTrainDim*dTrainDim))*derv_i; //novo nome

            //dyi/dw = derv_ActFn_t * x_t
            for (unsigned int in=0; in<DimIn; in++)
            [
                double momterm = momRate * ( Weights1(in,hid) - prev_Weights1(in,hid)); //momentum term
                delta_weights1(in,hid) = ( MM * (-1.0) * lrCoeff1 * (error_i * TrainSet(i,in)) ) + momterm ; //MaxE = Min IP
                prev_Weights1(in,hid) = Weights1(in,hid);
                Weights1(in,hid) += delta_weights1(in,hid);
            ]
            prevBias1(hid) = Bias1(hid);
            Bias1(hid) += ( MM * (-1.0) * lrCoeff1 * (error_i) ) + ( momRate * ( Bias1(hid) - prevBias1(hid) ) );
        ]
    ]
]
```

MaxMI_CS sub-code

Before entering within the loop over the epochs for the first half, it is necessary to perform some calculations. According to equations for information forces and cost function calculation eq. R5/(30), eq. R5/(32), eq. R5/(33), eq. R5/(34) and eq. R5/(37), we need to calculate the quantities V_{ij}^x, V_{ij}^y, V_i^x, V_i^y, V^x, V^y and d_{ij}, where x is the input instance and y is the resulting instance in hidden layer. As V_{ij}^x, V_i^x, V^x only depend on input values, its calculation is performed separately.

The code starts to declare the matrix **Vij_x,** that will store the Gaussian $G(y_i - y_j, 2\sigma^2 I)$, as defined in equation eq. R5/(26). Next, the vectors **dij** and **Vi_x** are declared (these vectores will store the divergence between the instances under assessment and the average of Gaussians, according to eq. R5/(27), respectively). The code proceeds with the initialization of the constants **aux_vix**, **aux_vx** (auxiliary parameters for the construction of **Vi_x** and **V_x**) and **V_x** (that will store the quantity defined in equation eq. R5/(28)).

The code proceeds with the initialization of three loops. The first two loops evolve over TrainDim (**i** and **j,** referring to the instances dimension). The third loop evolves over DimIn (**m** referring to the input dimention**)**. Inside the third loop, the vector **dij** is calculated according to the definition ${d_{ij}}^x = x_i - x_j$.

Vij_x is calculated based on *Vij_function* function, for the cross of instances. **aux_vix** is the sum of all component of matrix **Vij_x**. Only considering first loop, **Vi_x** is calculated based on **aux_vix**, which is restarted every loop. **Aux_vx** is defined as the sum of all components of **Vi_x**. Outside loops, **V_x** is calculated based on **aux_vx**.

```
// ##### Calculus Vij_inputs #####

mat Vij_x(TrainDim,TrainDim);

vec dij(DimIn), Vi_x(TrainDim);

double aux_vix = 0.0, aux_vx= 0.0, V_x = 0.0;


for (int i=0; i<TrainDim; i++)

[

for (int j=0; j<TrainDim; j++)

   [

      for (int in=0; in<DimIn; in++)

      [dij(in) = TrainSet(i,in) - TrainSet(j,in);]

      //dij = TrainSet.row(i) - TrainSet.row(j);

      Vij_x(i,j) = Vij_function(dij, DimIn, sig2);

      aux_vix += Vij_x(i,j);

   ]
```

```
    Vi_x(i) = aux_vix/dTrainDim;
    aux_vix = 0.0;
    aux_vx += Vi_x(i);
]
V_x = aux_vx/dTrainDim;
// ##### End Calculus Vij_inputs #####
```

Next, the sub-code for MaxMI_CS, inside the loop over Epochs1. Declaration of matrix **Vij_y**; vectors **dij_y** and **Vi_y**; double **aux_viy**. That variables' calculations are similar to **Vij_x**, **dij_y**, **Vi_x** and **aux_vix** but for neurons at hidden layer, $\boldsymbol{y}$.

```
// ##### Calculus of Vij concerning the hidden layer #####
mat Vij_y(TrainDim,TrainDim);
vec dij_y(DimHidden), Vi_y(TrainDim);
double aux_viy = 0.0;
for (int i=0; i<TrainDim; i++)
[
    for (int j=0; j<TrainDim; j++)
    [
        for (int hid=0; hid<DimHidden; hid++)
        [
            dij_y(hid) = HiddenTrain(i,hid) - HiddenTrain(j,hid);
        ]
        Vij_y(i,j) = Vij_function(dij_y, DimHidden, sig2);
        aux_viy += Vij_y(i,j);
    ]
    Vi_y(i) = aux_viy/dTrainDim;    aux_viy = 0.0;
]
```

// ##### End of Calculus of Vij concerning the hidden layer #####

To calculate the information force quantity in equation eq. R5/(35), we divide the calculation by steps according next formula:

$$F_i = -\frac{1}{N^2\sigma^2}\left[\frac{N1}{D1} + \frac{N2}{D2} - 2\frac{N3}{D3}\right], \qquad \boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j \qquad \textbf{eq. R5/(51)}$$

Where,

$$N1 = \sum_j V_{ij}^x V_{ij}^y \boldsymbol{d}_{ij}\,, \quad N2 = \sum_j V_{ij}^y \boldsymbol{d}_{ij}\,, \quad N3 = \sum_j V_j^x V_{ij}^y \boldsymbol{d}_{ij} \qquad \textbf{eq. R5/(52)}$$

$$D1 = \sum_i \sum_j V_{ij}^x V_{ij}^y\,, \quad D2 = \sum_i \sum_j V_{ij}^y\,, \quad D3 = \sum_j V_j^x V_j^y \qquad \textbf{eq. R5/(53)}$$

In next code block, "*Auxiliares no Calculo Parcela 1 e 2: D1 e D2*", declaration and calculation of **aux_vj** and **aux_vm** auxiliary variables are done. These variables will help the **D1** and **D2** calculation.

```
// ##### Auxiliares no Calculo Parcela 1 e 2: D1 e D2 #####
double aux_vj = 0.0, aux_vm = 0.0;
for (int i=0; i<TrainDim; i++)
[
for (int j=0; j<TrainDim; j++)
   [
      aux_vj += Vij_x(i,j) * Vij_y(i,j);
      aux_vm += Vij_y(i,j);
   ]
]
// ##### Fim Auxiliares no Calculo Parcela 1 e 2: D1 e D2 #####
```

Inside "BackForward Step == MaxMI_Cauchy-Swartz Criteria" Block there are the declarations of next variables: vectors **y_i**, **y_j**, **Fi**, **dij**, **auxN1**, **auxN2**, **auxN3**, **N1**, **N2**, **N3**; doubles **Vij_out**, **D1**, **D2**, **D3**. **y_i** and **y_j** will store output neurons for instance **i** e **j**, respectively. **dij** is the divergence $\boldsymbol{d}_{ij} = \boldsymbol{y}_i - \boldsymbol{y}_j$, **Fi** will store

the information force according to eq. R5/(51). **aux_N1**, **aux_N2**, **aux_N3** will help on **N1**, **N2** and **N3** calculations according to eq. R5/(52). Declaration of matrix **delta_weights1** inside instance **j** loop. This matrix will store the variation on weights (for the first part of the autoencoder). Next loop, **k**, we have the vector **Fi** calculation according to eq. R5/(51). Then, according to eq. R5/(11) and explanation above (at **MaxE sub-code**), we derive update for weights and bias at first part of autoencoder. This method is equivalent to maximize Information Potential. As we want maximize, learning rate should be positive. As **lrCoeff1** was defined in the beginning of code as a positive parameter, we multiply the **delta_weights1** formula by constant (1.0).

```
//##### BackForward Step == MaxMI_Cauchy-Swartz Criterium #####

vec y_i(DimHidden), y_j(DimHidden), Fi(DimHidden);

vec dij(DimHidden), auxN1(DimHidden), auxN2(DimHidden), auxN3(DimHidden), N1(DimHidden), N2(DimHidden), N3(DimHidden);

double Vij_out;

double D1=0.0, D2=0.0, D3=0.0;

D1 = aux_vj;

D2 = aux_vm;

N1.zeros(), N2.zeros(), N3.zeros();


for (unsigned int i=0; i<TrainDim; i++) //Cross samples: sample t

[

   for (unsigned int j=0; j<TrainDim; j++) //Cross samples: sample s

   [

      for(unsigned int hid=0; hid<DimHidden; hid++) //for each output neuron

      [

         y_i(hid) = HiddenTrain(i,hid);

         y_j(hid) = HiddenTrain(j,hid);

         dij(hid) = y_i(hid) - y_j(hid);

         // ############ dMI/dyi = Fi
```

```
        // Principe Tutorial pp.25 Eq.28
        auxN1(hid) = Vij_x(i,j) * Vij_y(i,j) * dij(hid);
        auxN2(hid)= Vij_y(i,j) * dij(hid);
        auxN3(hid) = Vi_x(j) * Vij_y(i,j) * dij(hid);
      ]
      N1 += auxN1;
      N2 += auxN2;
      N3 += auxN3;
      D3 += Vi_x(j) * Vi_y(j);
   ]
   // ############ dMI/Dw = dMI/dy * dy/dw ############
   mat delta_weights1(DimIn,DimHidden);
   for(unsigned int hid=0; hid<DimHidden; hid++) //for each output neuron: derivative
   [
      Fi(hid) = -1.0/( dTrainDim*dTrainDim * sig2 ) * ( N1(hid)/D1 + N2(hid)/D2 - 2.0*N3(hid)/D3 );
      //Derivative of ACTIVATION FUNCTION @ Y_s
      double derv_i = DervActFcn(y_i(hid), actFcnFlatness);
      //dMI/dyi = Fi
      //dy/dw = derv_ActFn_t * x_t - derv_ActFn_s * x_s
      for (unsigned int in=0; in<DimIn; in++)
      [
         double momterm = momRate * ( Weights1(in,hid) - prev_Weights1(in,hid)); //momentum term
         delta_weights1(in,hid) = ( MM * (1.0) * lrCoeff1 * Fi(hid) * (derv_i * TrainSet(i,in) ) ) + momterm
;//MaxMI = Max (-log IP) = Min log IP

         prev_Weights1(in,hid) = Weights1(in,hid);
```

```
        Weights1(in,hid) += delta_weights1(in,hid);
      ]
      prevBias1(hid) = Bias1(hid);
      Bias1(hid) += ( MM * (1.0) * lrCoeff1 * Fi(hid) * derv_i ) + ( momRate * ( Bias1(hid) - prevBias1(hid) ) );
    ]
]
```

//##### End of BackForward Step == MaxMI_Cauchy-Swartz Criterium #####

MaxMI_ED sub-code

Similarly to the CS method, the quantities C_{ij}^{x}, V_{ij}^{x}, V_{ij}^{y}, V_{i}^{x}, V_{i}^{y}, V^{x}, V^{y} and $\boldsymbol{d}_{ij}$ are calculated following eq. R5/(39), eq. R5/(40) and eq. R5/(42), where x is an input instance and y is an hidden instance. As the quantities C_{ij}^{x}, V_{ij}^{x}, V_{i}^{x}, V^{x} depend only on the input values, the calculation of these measures is performed outside the loop over epochs 1. Declaration of matrix **Cij_x** that will store $C_{ij}^{x} = V_{ij}^{x} - 2V_{i}^{x} + V^{x}$, by definition; vectors **xi**, **xj** and **dij** that will store input instance **i** e **j** and the divergence between the instances under assessment, respectively; The variables (double type) **V**, **Vi**, **Vj**, **Vij** are calculated according to eq. R5/(26), eq. R5/(27) and eq. R5/(28), **V** is calculated based on *V_function*; **Vj** is calculated based on *Vi_function*; **Vij** is calculated based on *Vij_function*; **Vi** is calculated based on *Vi_function*; finally, **Cij** is calculated based on its definition.

//##### Cij Calculation #####

```
// CALCULATE c_ij for input instances s and t
mat Cij(TrainDim, TrainDim);
double V, Vi, Vj, Vij;
vec xi(DimIn), xj(DimIn), dij(DimIn);

V = V_function(TrainSet, DimIn, sig2, TrainDim);

for(unsigned int i=0; i< TrainDim; i++)
[
```

```
for(unsigned int in=0; in<DimIn; in++)
[xj(in) = TrainSet(i,in);//input of instance t]

Vj = Vi_function(TrainSet, xj, DimIn, sig2, TrainDim);

for (unsigned int j=0; j< TrainDim; j++)
[
   for(unsigned int in=0; in<DimIn; in++)
   [
      xi(in) = TrainSet(j,in);//input of instance s
      dij(in) = xi(in) - xj(in);
   ]
   Vij = Vij_function(dij, DimIn, sig2);
   Vi = Vi_function(TrainSet, xi, DimIn, sig2, TrainDim);
   Cij(i,j) = Vij - Vi - Vj + V;
]
]
//##### End of Cij Calculation #####
```

Sub-code for MaxMI_ED, inside epochs loop, starts with declaration of vectors **y_i**, **y_j** and **dij2**; double **Vij_out**; matrices **delta_weights1** and **aux_delta**. Again, vectors **y_i**, and **y_j** will store the instances **i** and **j** under assessment, and **dij2** that will store the divergence between vectors bellow. **Vij_out**, calculated based on *Vij_function*, will store the value of the Gaussian of the output neurons in hidden layer, $G(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I)$ and matrix **aux_deltas** will store the value of the total variation in weights (at first part of autoencoder) into that moment. Next part of sub-code is similar to correspondent part of MaxE method. New loop (unsigned int **k**) has implemented where, for each output neuron, the corresponding new weights and bias will be calculated. For that, derivatives of activation function for instances **i** and **j** will be stored at variables **derv_i** and **derv_j** based on *DervActFcn* function. From eq. R5/(40) and eq. R5/(41):

$$\frac{\partial}{\partial \omega} I_{ED}(X,Y) = \sum_{i=1}^{N} [F_i]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega}$$

$$[F_i]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} = \left(-\frac{1}{N^2\sigma^2} \sum_{j=1}^{N} C_{ij}^x V_{ij}^y \boldsymbol{d}_{ij} \right) \cdot \left(\frac{\partial \boldsymbol{y}_i}{\partial net_i} \frac{\partial net_i}{\partial \omega} \right)$$

So, in analogy with theory bellow,

$\mathbf{derv\_s} = \frac{\partial \boldsymbol{y}_s}{\partial net_s}$ and $\mathbf{derv\_t} = \frac{\partial \boldsymbol{y}_t}{\partial net_t}$

$$\mathbf{error\_s} = F_s \cdot \frac{\partial \boldsymbol{y}_s}{\partial net_s} = \left(-\frac{1}{N^2\sigma^2} \sum_{j=1}^{N} C_{ij}^x V_{ij}^y \boldsymbol{d}_{ij} \right) \cdot \mathbf{derv\_s}$$

$$\mathbf{error\_t} = F_t \cdot \frac{\partial \boldsymbol{y}_t}{\partial net_t} = \left(-\frac{1}{N^2\sigma^2} \sum_{j=1}^{N} C_{ij}^x V_{ij}^y \boldsymbol{d}_{ij} \right) \cdot \mathbf{derv\_t}$$

Loop at **unsigned int i** variable, identifies each weight by the output neuron (**unsigned int k**) and input neuron (**unsigned int i**). For each weight in this loop, momentum term is calculated (despite **momRate** parameter defined equal to zero). Then **delta_weights1** is calculated according to eq. R5/(3):

$$\mathbf{delta\_weights1} = \Delta\omega_{ij} = \eta \frac{\partial V}{\partial \omega_{ij}} + momterm$$

From equations bellow, lack calculate $\frac{\partial net_s}{\partial w_{si}}$. From definition of net_s, in eq. R5/(49), follows that:

$\frac{\partial net_s}{\partial w_{si}} = \boldsymbol{a}_i = \mathbf{TrainSet(t,i)}$.

So,

$$\mathbf{delta\_weights1} = \eta \frac{\partial V}{\partial \omega_{ij}} + momterm$$

$$= \eta \sum_{i=1}^{N} [F_i]^T \frac{\partial \boldsymbol{y}_i}{\partial \omega} + momterm$$

$$\mathbf{delta\_weights1(i,k)} = \eta \cdot F_t \cdot \left(\frac{\partial \boldsymbol{y}_t}{\partial net_t} \frac{\partial net_t}{\partial \omega} \right) + momterm$$

$$= \eta \cdot \mathbf{error\_t} \cdot \frac{\partial net_t}{\partial \omega} + momterm$$

$$= \eta \cdot \mathbf{error\_t} \cdot \mathbf{TrainSet(t,i)} + momterm$$

In same loop, for each weight, **aux_deltas** (a control variable) is updated as the sum of latest **delta_weights1**; **prev_Weights1** is set equal to the actual weights (from the last epoch) and **Weights1** is a matrix of new weights that sums to the latest weights their variations (**delta_weights1**). All these variables concern to the weights at first part of autoencoder. Outside of this loop, i.e., only considering the loop in **k** (output neurons), identical process is done for update bias value. The variable **prevBias1** stores the last bias (previous epoch) and variable **Bias** sums to the old bias component the new bias variation:

$$\mathbf{Bias1}^{(r)} = \mathbf{Bias1}^{(r-1)} + \Delta b_j$$

(At code, this equation is equivalent to $\mathbf{Bias1} \mathrel{+}= \Delta b_j$)

$$\Delta b_j = \eta \frac{\partial V}{\partial b_j} + momterm = \eta \cdot \mathbf{error\_t} + momterm$$

This method is equivalent to maximize Information Potential. As we want to maximize, the learning rate should be positive. As **lrCoeff1** is defined in the beginning of code as a positive parameter, we multiply the **delta_weights1** formula by constant (1.0).

```
//##### BackForward Step == MaxMI Criterium #####
vec y_i(DimHidden), y_j(DimHidden), dij2(DimHidden);
double Vij_hid;
mat delta_weights1(DimIn,DimHidden);

for (unsigned int i=0; i<TrainDim; i++) //Cross samples: sample i
[
   for (unsigned int j=0; j<TrainDim; j++) //Cross samples: sample j
   [
      for(unsigned int hid=0; hid<DimHidden; hid++) //for each output neuron
      [
         y_i(hid) = HiddenTrain(i,hid);
         y_j(hid) = HiddenTrain(j,hid);
         dij2(hid) = y_i(hid) - y_j(hid);
      ]
```

```
        Vij_hid = Vij_function(dij2, DimHidden, sig2); // Vij = G(yi-yj, 2sig2)


        // dV/Dw = dV/dy * dy/dw
        for(unsigned int hid=0; hid<DimHidden; hid++) //for each output neuron: derivative
        [
            // Act Function Derivative @ Y_s
            double derv_j = DervActFcn(y_j(hid), actFcnFlatness);
            double derv_i = DervActFcn(y_i(hid), actFcnFlatness);


            //dV/dy = dij[k] * Vij_out * (-1.0/(2.0*sig2*TrainDimIn*TrainDimIn))
            double error_j = dij2(hid) * Cij(i,j) * Vij_hid * (-1.0/(2.0*sig2*dTrainDim*dTrainDim))*derv_j;
            double error_i = dij2(hid) * Cij(i,j) * Vij_hid * (-1.0/(2.0*sig2*dTrainDim*dTrainDim))*derv_i;


            //dy/dw = derv_ActFn_t * x_t - derv_ActFn_s * x_s
            for (unsigned int in=0; in<DimIn; in++)
            [
                double momterm = momRate * ( Weights1(in,hid) - prev_Weights1(in,hid)); //momentum term
                delta_weights1(in,hid) = ( (1.0) * lrCoeff1 * (error_i * TrainSet(i,in)) ) + momterm ;
                prev_Weights1(in,hid) = Weights1(in,hid);
                Weights1(in,hid) += delta_weights1(in,hid);
            ]
            prevBias1(hid) = Bias1(hid);
            Bias1(hid) += ((1.0) * lrCoeff1 * (error_i)) + (momRate * (Bias1(hid) - prevBias1(hid)));
        ]
    ]
]
```

//##### End of BackForward Step == MaxMI Criterium #####

MinMI_CS_Uni sub-code

Before epochs loops, total_**CostFn** is declared and set to be equal to a big number (10000.0) in order to the Learning Coefficient Adaptive Criteria do not cut the parameter **lrcoeff1** at first iteration/epoch; **prev_total_CostFn** is declared and initialized equal to zero.

```
double total_CostFn = 10000.0, prev_total_CostFn = 0.0;
```

Sub-code for MinMI_CS_Uni starts with **delta_weights1** declaration, as above will store variation on weights and **total_CostFn** is now defined with value 0 and corresponds to quantity in eq. R5/(44). This method applies the same approach as above, on MaxMI_CS case, but for the CS univariate case. This means minimize MI between univariate variables, in that case, between each combination of 2 different neurons of hidden layer. To do that we will cross output neurons in hidden layer (loop **n** and **m**) to calculate the correspondent mutual information for this pair/combination. Second loop, m ranges between $(\mathrm{n}+1)$ and **DimHidden** because of MI symmetry. The variables used in this sub code are similar with the ones used on MaxMI_CS method. Instead x be the input vector and y the hidden vector, in this case, x is one neuron at the hidden layer and y is another one neuron at the hidden layer. For example, declaration of matrices **Vij_x** and **Vij_y** will store matrices V_{ij} (eq. R5/(26)) for active neuron x and neuron y in hidden layer; declaration of vectors **Vj_x** and **Vj_y** (eq. R5/(27)) will store vector V_i for active neuron x and neuron y in hidden layer; etc.

```
// ##### Calculo Vij_saidas #####

mat delta_weights1(DimIn,DimHidden);

delta_weights1.zeros();


total_CostFn = 0.0;


for (unsigned int n=0; n< DimHidden; n++)

[

   for (unsigned int m=n+1; m< DimHidden; m++)

   [
```

```
mat Vij_x(TrainDim,TrainDim), Vij_y(TrainDim,TrainDim);
vec Vj_x(TrainDim), Vj_y(TrainDim);
double aux_vjx = 0.0, aux_vjy = 0.0, dij_y = 0.0, dij_x = 0.0;
double aux_V_x = 0.0, aux_V_y = 0.0, V_x, V_y;

for (unsigned int i=0; i<TrainDim; i++)
[
   for (unsigned int j=0; j<TrainDim; j++)
   [
      dij_x = HiddenTrain(i,m) - HiddenTrain(j,m);
      dij_y = HiddenTrain(i,n) - HiddenTrain(j,n);

      Vij_x(i,j) = Vij_function(dij_x, 1, sig2);
      Vij_y(i,j) = Vij_function(dij_y, 1, sig2);

      aux_vjx += Vij_x(i,j);
      aux_vjy += Vij_y(i,j);
   ]
   Vj_x(i) = aux_vjx/dTrainDim;
   Vj_y(i) = aux_vjy/dTrainDim;

   aux_V_x += Vj_x(i);
   aux_V_y += Vj_y(i);

   aux_vjx = 0.0;
   aux_vjy = 0.0;
```

```
]
V_x = aux_V_x / dTrainDim;
V_y = aux_V_y / dTrainDim;

// ##### Fim Calculo Vij_saidas #####

// ##### Calculo Vj #####
double aux_vj = 0.0, aux_vmx = 0.0, aux_vmy = 0.0;
for (unsigned int i=0; i<TrainDim; i++)
[
   for (unsigned int j=0; j<TrainDim; j++)
   [
      aux_vj += Vij_x(i,j) * Vij_y(i,j);
      aux_vmx += Vij_x(i,j);
      aux_vmy += Vij_y(i,j);
   ]
]
// ##### Fim Calculo Vj #####

//##### BackForward Step == MinMI_Cauchy-Swartz Criteria #####
double x_t = 0.0, x_s = 0.0, y_t = 0.0, y_s = 0.0;
double Fi_x=0.0, Fi_y=0.0;
double N1_y=0.0, D1=0.0, N2_y=0.0, D2_x=0.0, D2_y=0.0, N3_y=0.0, D3=0.0;
double N1_x=0.0, N2_x=0.0, N3_x=0.0;
D1 = aux_vj;
D2_x = aux_vmx;
```

```
D2_y = aux_vmy;

dij_y = 0.0, dij_x = 0.0;


for (unsigned int i=0; i<TrainDim; i++) //Cross samples: sample i
[
   for (unsigned int j=0; j<TrainDim; j++) //Cross samples: sample j
   [
      x_t = HiddenTrain(i,m);//neuronio saida 1 patt i
      x_s = HiddenTrain(j,m);//neuronio saida 1 patt j
      y_t = HiddenTrain(i,n);//neuronio saida 2 patt i
      y_s = HiddenTrain(j,n);//neuronio saida 2 patt j


      dij_x = x_t - x_s;
      dij_y = y_t - y_s;


      // ############# dMI/dyi = Fi
      // ver Principe Tutorial pag.25 Eq.28
      N1_x += Vij_x(i,j) * Vij_y(i,j) * dij_x;
      N2_x += Vij_x(i,j) * dij_x;
      N3_x += Vj_x(j) * Vij_y(i,j) * dij_x;


      N1_y += Vij_x(i,j) * Vij_y(i,j) * dij_y;
      N2_y += Vij_y(i,j) * dij_y;
      N3_y += Vj_x(j) * Vij_y(i,j) * dij_y;
      D3 += Vj_x(j) * Vj_y(j);
   ]
```

```
Fi_x = -1.0/( dTrainDim*dTrainDim * sig2)*( N1_x/D1 + N2_x/D2_x - 2.0*N3_x/D3 );

Fi_y = -1.0/( dTrainDim*dTrainDim * sig2)*( N1_y/D1 + N2_y/D2_y - 2.0*N3_y/D3 );


// ############# dMI/Dw = dMI/dy * dy/dw #############


// Act Function Derivative @ Y_s

//double derv_s = DervActFcn(y_s, actFcnFlatness);

double dervx_t = DervActFcn(x_t, actFcnFlatness);

double dervy_t = DervActFcn(y_t, actFcnFlatness);


//dMI/dyi = Fi

//dy/dw = derv_ActFn_t * x_t - derv_ActFn_s * x_s

for (unsigned int in=0; in<DimIn; in++)

[

   double momterm_x = momRate * ( Weights1(in,m) - prev_Weights1(in,m)); //momentum term

   double momterm_y = momRate * ( Weights1(in,n) - prev_Weights1(in,n)); //momentum term


   delta_weights1(in,m) = ( MM * (1.0) * lrCoeff1 * Fi_x * (dervx_t * TrainSet(i,in) ) ) + momterm_x ;
//Min MI entre neuronios = Min (-log(V(y)) = Max log V(y)

   delta_weights1(in,n) = ( MM * (1.0) * lrCoeff1 * Fi_y * (dervy_t * TrainSet(i,in) ) ) + momterm_y ;


   prev_Weights1(in,m) = Weights1(in,m);

   Weights1(in,m) += delta_weights1(in,m);


   prev_Weights1(in,n) = Weights1(in,n);

   Weights1(in,n) += delta_weights1(in,n);
```

```
        ]
        prevBias1(m) = Bias1(m);
        prevBias1(n) = Bias1(n);

        Bias1(m) += ( MM * (1.0) * lrCoeff1 * Fi_x * dervx_t ) + ( momRate * ( Bias1(m) – prevBias1(m) ) );
        Bias1(n) += ( MM * (1.0) * lrCoeff1 * Fi_y * dervy_t ) + ( momRate * ( Bias1(n) – prevBias1(n) ) );
      ]
   ]
]
//##### End of BackForward Step == MinMI_ED Criteria #####
```

Output matrix

When all instances (**TrainDim** instances) are "traveled", and the respective cumulated change on weights are complete (updated **Weights1**), a new output matrix is calculated, using the last changes performed on the weights (**Weights1**).

```
//##### Update Hidden Matrix == FeedForward Step #####
mat prev_HiddenTrain (TrainDim, DimHidden);
prev_HiddenTrain = HiddenTrain;
double aux1 = 0.0;

for (unsigned int tr=0; tr<TrainDim; tr++)
[
   for(unsigned int hid=0; hid<DimHidden; hid++)
   [
      for(unsigned int in=0; in<DimIn; in++)
      [aux1 += ((TrainSet(tr,in))*Weights1(in,hid));]
      HiddenTrain(tr,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness);
      aux1 = 0.0;
```

]

]

//##### End of Update Hidden Matrix #####

Saturation Process

One way to avoid the network saturation is by randomly perturbation of neurons. This was made by introducing a random perturbation in the weights. Next code implements this perturbation. Definition of matrix **aux_saturation1** that identifies saturated elements of matrix *HiddenTrain* based on saturation limit (**SatLim**). If an element in *HiddenTrain* matrix is higher, in modulus, that the saturation limit then **aux_saturation1** stores in the corresponding place 1. For the non saturated elements stores 0. Definition of variable **aux_SatLevel1** that sums all the elements of matrix **aux_saturation1,** giving the total of its saturated elements. **SatLevel1** is the percentage of saturated elements. If the network is saturated, this is, if **SatLevel1** is higher than a threshold value (**ThresholdSat**) then weights will be pertubated according to:

$$W_{new} = (random * W) + W \qquad \textbf{eq. R5/(54)}$$

The $random$ number is defined as $0.1 * \text{aleat(rng)}$, where aleat(rng) generates uniform real numbers between the interval $[-1, 1]$, so $random$ is a uniform number between $[-0.1, 0.1]$. This means that weights suffer a perturbation of 10% into both directions.

//######## SATURATION process #####

mat aux_saturation(TrainDim,DimHidden); //auxiliary matrix to calculate the proportion of weigths within saturation state (the weights the absoulte value higher than 0.9)

aux_saturation.zeros();

for (unsigned int tr=0; tr<TrainDim; tr++)

[

for(unsigned int hid=0; hid<DimHidden;hid++)

[

if (abs(HiddenTrain(tr,hid))>=SatLimit)

[

```
// the matrix with "1" for the corresponding positions of saturated weights (for an absolute specified value, e.g. 0.9)
            aux_saturation(tr,hid) =1;
        ]
        else
        [aux_saturation(tr,hid) =0;]
    ]
]
double aux_SatLevel=accu(aux_saturation); //retorna a soma acumulada de todos os elementos que estão contidos dentro da matrix auxiliar
double SatLevel=aux_SatLevel/(dTrainDim*dDimHidden); //retorna a percentagem de weights 1 que estão saturados

if(SatLevel>ThresholdSat)
[
    for(unsigned int hid=0; hid<DimHidden;hid++)
    [
        for (unsigned int in=0; in<DimIn; in++)
        [Weights1(in,hid) =  0.1*aleat(rng)*Weights1(in,hid)+Weights1(in,hid);]
        Bias1(hid) = 0.1*aleat(rng)*Bias1(hid)+Bias1(hid);
    ]
    aux1 = 0.0;
    for (unsigned int tr=0; tr<TrainDim; tr++)
    [
        for(unsigned int hid=0; hid<DimHidden;hid++)
        [
```

```
        for(unsigned int in=0; in<DimIn;in++)
        [aux1 += ((TrainSet(tr,in))*Weights1(in,hid));]
        HiddenTrain(tr,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness);
        aux1 = 0.0;
      ]
   ]
]
//######## END SATURATION process #####
```

Cost function

The goal of this work is training a neural network maximizing or minimizing some cost functions. Next, there are the cost functions for the different ITL methods.

MaxE Cost Function

For this method, the goal is maximize the entropy. So **Entropy**, according to eq. R5/(14), is the cost function.

$$CostFunction = -\log\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} G\left(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I\right)\right) \qquad \textbf{eq. R5/(55)}$$

The code is the direct calculation of this formula, where $\mathbf{dij3} = \boldsymbol{y}_i - \boldsymbol{y}_j$ and $\mathbf{tFn} = \sum_{i=1}^{N}\sum_{j=1}^{N} G\left(\boldsymbol{y}_i - \boldsymbol{y}_j, 2\sigma^2 I\right)$.

```
//##### Calculating Cost Function for this Hidden Matrix #####
double aux_CostFn = 0.0;
vec dij3(DimHidden);
for (unsigned int i=0; i<TrainDim; i++)
[
   for (unsigned int j=0; j< TrainDim; j++)
   [
      for (unsigned int hid=0; hid< DimHidden; hid++)
```

```
        [dij3(hid) = HiddenTrain(i,hid) - HiddenTrain(j,hid); ]

        aux_CostFn += Vij_function(dij3, DimHidden, sig2);

    ]

]

Prev_CostFn = CostFn;

CostFn = -log10 (( 1.0 / (pow(dTrainDim,2.0)) ) * aux_CostFn);

//##### End of Calculating Cost Function for this Hidden Matrix #####
```

MaxMI_CS Cost Function

The cost function for this method is the Cauchy-Schwartz Mutual Information estimation, according eq. R5/(37). In that way,

$$CostFunction = \log\frac{\left(\frac{1}{N^2}\sum_i\sum_j V_{ij}^x V_{ij}^y\right)(V^x V^y)}{\left(\frac{1}{N}\sum_i V_i^x V_i^y\right)^2}$$
$$= \log\left(\frac{1}{N^2}\sum_i\sum_j V_{ij}^x V_{ij}^y\right) + \log(V^x V^y) - 2 * \log\left(\frac{1}{N}\sum_i V_i^x V_i^y\right) \quad \textbf{eq. R5/(56)}$$

Where,

$$\mathbf{dij\_CF\_y} = y_i - y_j = \mathbf{HiddenTrain}(i,k) - \mathbf{HiddenTrain}(j,k)$$

$$\boldsymbol{Vij\_CF\_y} = V_{ij}^y$$

$$\boldsymbol{Vi\_CF\_y} = V_i^y$$

$$\boldsymbol{V\_CF\_y} = V^y$$

$$\boldsymbol{VJ\_CF} = \frac{1}{N^2}\sum_i\sum_j V_{ij}^x V_{ij}^y$$

$$\boldsymbol{VM\_CF} = V^x V^y$$

$$\boldsymbol{VC\_CF} = \left(\frac{1}{N}\sum_i V_i^x V_i^y\right)$$

So, cost function becomes:

$$CostFunction = \log(\boldsymbol{VJ\_CF}) + \log(\boldsymbol{VM\_CF}) - 2 * \log(\boldsymbol{VC\_CF}) \quad \textbf{eq. R5/(57)}$$

```
//########## Calculating Cost Function for this Output ##########
double aux_CostFn = 0.0, aux_Vi_CF_y = 0.0, aux_V_CF_y = 0.0, V_CF_y = 0.0;
mat Vij_CF_y(TrainDim,TrainDim);
vec Vi_CF_y(TrainDim);
vec dij_CF_y(DimHidden);

for (unsigned int i=0; i<TrainDim; i++)
[
   for (unsigned int j=0; j<TrainDim; j++)
   [
      for (unsigned int hid=0; hid<DimHidden; hid++)
      [dij_CF_y(hid) = HiddenTrain(i,hid) - HiddenTrain(j,hid);]
      Vij_CF_y(i,j) = Vij_function(dij_CF_y, DimHidden, sig2);
      aux_Vi_CF_y += Vij_CF_y(i,j);
   ]
   Vi_CF_y(i) = aux_Vi_CF_y / dTrainDim;
   aux_V_CF_y += Vi_CF_y(i);
]
V_CF_y = aux_V_CF_y / dTrainDim;
double aux_VJ_CF = 0.0, aux_VC_CF = 0.0;
double VJ_CF=0.0, VM_CF=0.0, VC_CF=0.0; //the 3 parts of the cost function: joint, marginal and cross information potentials
for (int i=0; i<TrainDim; i++)
```

```
[
   for (int j=0; j<TrainDim; j++)
   [aux_VJ_CF += Vij_x(i,j) * Vij_CF_y(i,j);]
   aux_VC_CF += Vi_x(i) * Vi_CF_y(i);
]
VJ_CF = aux_VJ_CF / ( dTrainDim*dTrainDim ); //the argument for the joint information potential
aux_VJ_CF = 0.0;
VM_CF = V_x * V_CF_y; //the argument for the marginal information potential
VC_CF = aux_VC_CF / ( dTrainDim ); //the argunment for the cross information potential
aux_VC_CF = 0.0;
Prev_CostFn = CostFn;
CostFn = (log(VJ_CF) + log(VM_CF) - 2.0*log(VC_CF)); //the complete information potential
//########## End of Calculating Cost Function for this Output ##########
```

MaxMI_ED Cost Function

For Euclidean Distance Mutual Information estimation, the cost function according to eq. R5/(43), is:

$$CostFunction = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij}^{x}\ V_{ij}^{y} \qquad \textbf{eq. R5/(58)}$$

Where C_{ij}^{x} was calculated before epochs loop and $V_{ij}^{y} = \mathbf{Vij\_function}(\mathrm{dij3}, \mathrm{DimHidden}, \mathrm{sig2})$ is calculated with updated variable $\mathbf{dij3}$ (using the new output matrix **HiddenTrain**)

```
//##### Calculating Cost Function for this Output #####
double aux_CostFn = 0.0;
vec dij3(DimHidden);
for (unsigned int i=0; i<TrainDim; i++)
[
   for (unsigned int j=0; j< TrainDim; j++)
```

```
    [
        for (unsigned int hid=0; hid<DimHidden; hid++)
        [dij3(hid) = HiddenTrain(i,hid) - HiddenTrain(j,hid); ]
        aux_CostFn += Cij(i,j) * Vij_function(dij3, DimHidden, sig2);
    ]
]
Prev_CostFn = CostFn;
CostFn = ( 1.0 / (pow(dTrainDim,2.0)) ) * aux_CostFn;
//##### End of Calculating Cost Function for this Output #####
```

MinMI_CS_Uni Cost Function

For this method, the cost function is the same as the MaxMI_CS. But in that case, MI is calculated between neurons of the hidden layer, instead input vector and hidden vector as in case of method MaxMI_CS. To do that, same adjustments need to be done. The quantities V_{ij}^{x}, V_{ij}^{y}, V_{i}^{x}, V_{i}^{y}, V^{x}, V^{y} and d_{ij} corresponds now to the combination of neurons. This is, $x = neuron\ n$ and $y = neuron\ m$. For this reason, these quantities will be calculated inside loops **n** and **m** (neurons combinations) based on updated neurons (**HiddenTrain**). "Calculating Cost Function for this HiddenMatrix" block corresponds to the direct calculation of cost function presented in eq. R5/(57), in MaxMIN_CS for each combination of neurons in hidden layer. In this method cost function will correspond to the summation of each MI of combination of neurons, the **total_CostFn** that want to be minimized, see eq. R5/(43).

$$CostFunction_i = \log(\boldsymbol{VJ\_CF}) + \log(\boldsymbol{VM\_CF}) - 2 * \log(\boldsymbol{VC\_CF}) \qquad \textbf{eq. R5/(59)}$$

$$TotalCostFunction = \sum_{i=1}^{\frac{L}{2}(L-1)} CostFunction_i \qquad \textbf{eq. R5/(60)}$$

```
prev_total_CostFn = total_CostFn;
//########## Calculating Cost Function for this HiddenMatrix ##########
for (unsigned int n=0; n< DimHidden; n++)
```

```
[
    for (unsigned int m=n+1; m< DimHidden; m++)
    [
        mat Vij_x(TrainDim,TrainDim), Vij_y(TrainDim,TrainDim);
        vec Vj_x(TrainDim), Vj_y(TrainDim);
        double aux_vjx = 0.0, aux_vjy = 0.0, dij_y = 0.0, dij_x = 0.0;
        double aux_V_x = 0.0, aux_V_y = 0.0, V_x, V_y;

        for (unsigned int i=0; i<TrainDim; i++)
        [
            for (unsigned int j=0; j<TrainDim; j++)
            [
                dij_x = HiddenTrain(i,m) - HiddenTrain(j,m);
                dij_y = HiddenTrain(i,n) - HiddenTrain(j,n);

                Vij_x(i,j) = Vij_function(dij_x, 1, sig2);
                Vij_y(i,j) = Vij_function(dij_y, 1, sig2);

                aux_vjx += Vij_x(i,j);
                aux_vjy += Vij_y(i,j);
            ]
            Vj_x(i) = aux_vjx/dTrainDim;
            Vj_y(i) = aux_vjy/dTrainDim;

            aux_V_x += Vj_x(i);
            aux_V_y += Vj_y(i);
```

```
    aux_vjx = 0.0;
    aux_vjy = 0.0;
  ]
  V_x = aux_V_x / dTrainDim;
  V_y = aux_V_y / dTrainDim;
  // ##### Fim Calculo Vij_saidas já com update #####

  //##### Calculating Cost Function for this HiddenMatrix #####
  double aux_VJ_CF = 0.0, aux_VC_CF = 0.0, CostFn = 0.0;
  double VJ_CF=0.0, VM_CF=0.0, VC_CF=0.0; //the 3 parts of the cost function: joint, marginal and cross information potentials

  for (int i=0; i<TrainDim; i++)
  [
    for (int j=0; j<TrainDim; j++)
    [
      aux_VJ_CF += Vij_x(i,j) * Vij_y(i,j);
    ]
    aux_VC_CF += Vj_x(i) * Vj_y(i);
  ]
  VJ_CF = aux_VJ_CF / ( dTrainDim*dTrainDim ); //the argument for the joint information potential
  aux_VJ_CF = 0.0;
  VM_CF = V_x * V_y; //the argument for the marginal information potential
  VC_CF = aux_VC_CF / ( dTrainDim ); //the argunment for the cross information potential
  aux_VC_CF = 0.0;
```

```
        CostFn = (log(VJ_CF) + log(VM_CF) - 2.0*log(VC_CF)); //the complete information potential
        //##### End of Calculating Cost Function for this Output #####
        total_CostFn += CostFn;
    ]
]
```

Stop criteria

At this point, the new value of the cost function is calculated (i.e. the real entropy or mutual information value at the end of each epoch). The model includes a stop criterion, which is activated when the magnitude increment of the cost function is really low (in this case, when the value of the cost function in the current epoch is lower than **goal** parameter. Note that this criterion is only active when the model is minimizing the objective function.

```
//##### Stop Criterium #####
//if the cost function do not evolve more than the criteria, than the loop of the epochs is closed
if ( abs(CostFn) < goal)
[Epochs1 = numEpochs1;]
//##### End Stop Criterium  #####
```

Adaptive learning coefficient

The model includes the adaptive evolution of the learning coefficient. The adaptive criterion is activated when the cost function evolves in the opposite direction to that expected. The implementation of this evaluation is made with the comparison of cost function values obtained in the previous and current epochs (using the ratio between the current cost function and the cost function obtained in the previous epoch, which is compared with 1 and with the variable ***max_perf_inc***), and taking in consideration the maximization or minimization objective of the problem.

◊ Maximization case

In the maximization case, the ratio ***CostFn/Prev_CostFn*** is expected to be higher than 1, meaning that the current epoch leaded the cost function to achieve higher values. This way, when the ratio is higher than 1,

the learning coefficient is increased by ***lr_inc***. For this case does not accept a value for learning rate higher than **Max_lrCoeff1**.

When the ratio ***CostFn/Prev_CostFn*** is lower than 1, the current epoch leaded to a decrease on the cost function. If this decrease is "not severe", the algorithm accepts the changes made in weights and bias during the current epoch and do not change the learning coefficient. The "not severe" is considered when the ratio ***CostFn/Prev_CostFn*** is comprised within the interval [1/max_perf_inc, 1]. If this decrease is "too severe", than the algorithm will reject the changes made in weight and bias during the current epoch, and will cut the learning rate by **lr_dec**. "Too severe" is considered when the ratio ***CostFn/Prev_CostFn*** is lower than *1/max_perf_inc*.

```
//##### Learning Coeff Adaptive #####
if( CostFn/Prev_CostFn < (1.0/max_perf_inc))
[
   lrCoeff1 = lrCoeff1 * lr_dec; //corta o learning rate
   Weights1 = prev_Weights1; //não aceita pesos
   Bias1 = prevBias1;
   HiddenTrain = prev_ HiddenTrain;
   if (lrCoeff1 < 0.00001)
   [
      lrCoeff1 = 0.00001;
   ]
]

if( CostFn/Prev_CostFn > 1.0)
[
   lrCoeff1 =  lrCoeff1 * lr_inc; //aceita pesos e aumenta learning rate
   if(lrCoeff1 > Max_lrCoeff1)
   [lrCoeff1 = Max_lrCoeff1;]
```

]

if((CostFn/Prev_CostFn >= (1.0/max_perf_inc)) && (CostFn/Prev_CostFn <= 1))

[

//aceita os pesos

//não altera o learning rate

]

//##### End Learning Coeff Adaptative #####

◊ Minimization case

When the cost function is a minimization, the reasoning made is the opposite. In this case, it is expected the ratio ***CostFn/Prev_CostFn*** to be lower than 1, which means that the current epoch allowed the achievement of a lower value to the cost function. This way, when the ratio ***CostFn/Prev_CostFn*** is lower than 1, the model accepts the changes in weigths and bias, and increases the learning rate (**lrCoeff2**) by **lr_inc**. This case does not accept a value for learning rate higher than **Max_lrCoeff1**.

When the ratio is higher than 1, two cases can occur: the "not severe" case or the "too severe" case, with similar rules to the ones exposed for the maximization case.

The "not severe" occurs when the ratio ***CostFn/Prev_CostFn*** is comprised between 1 and max_perf_inc. The "too severe" occurs when the ratio ***CostFn/Prev_CostFn*** is higher than max_perf_inc. In first case, accepts weights/bias and do not change the learning rate. In second case, do not accept weights/bias (use the previous ones) and cuts the learning rate by **lr_dec**. This second case do not accept **lrCoeff1** lower than 0.00001.

//##### Learning Coeff Adaptive #####

if(total_CostFn/prev_total_CostFn > max_perf_inc)

[

lrCoeff1 = lrCoeff1 * lr_dec; //corta o learning rate

Weights1 = prev_Weights1; //não aceita pesos

Bias1 = prevBias1;

HiddenTrain = prev_HiddenTrain;

```
    if (lrCoeff1 < 0.00001)
    [lrCoeff1 = 0.00001;]
]
if( total_CostFn/prev_total_CostFn < 1.0)
[
    lrCoeff1 =  lrCoeff1 * lr_inc; //aceita pesos e aumenta learning rate
    if(lrCoeff1 > Max_lrCoeff1)
    [lrCoeff1 = Max_lrCoeff1;]
]

if((total_CostFn/prev_total_CostFn >= 1.0) && (total_CostFn/prev_total_CostFn <= max_perf_inc))
[
    //aceita os pesos
    //não altera o learning rate
]
//##### End Learning Coeff Adaptative #####
```

At this point the epochs loop is complete.

Input information in csv files

After the loop including all epochs is complete, the model is now ready to provide the final results concerning the first part of the autoencoder. The results at this point can be evaluated with the cost function of the first part, which is the real value for entropy, or for mutual information. Therefore, the *csv file* is incremented with the following information at this part of the model: the initial value of learning coefficient 1, the final value of the learning coefficient 1 (the final value after running all epochs 1, which may include several cuts due to the adaptive procedure implemented), the seed for random numbers used in this experiment, and the final value of the cost function obtained in this experiment.

```
//Resultados_LrCoeff.open(name);
MaxE(v) = CostFn;
```

Resultados_LrCoeff << "lrCoeff1 Inicial:;" << learnCoeff[v] << ";lrCoeff1 Final:;" << lrCoeff1 << ";seed:;" << seed << ";MaxE:;" << MaxE(v) << ";";

//##### END OUTPUTs for each lrCoeff #####

Maximum learning coefficient to adopt in second part

Calculating the maximum value that can be adopted for the learning coefficient of the second part, the variable **Max_lrCoeff2**.

//###### maximum learnCoeff 2 value to be adopted #########

mat C2 (DimHidden, DimHidden);

C2 = cov(HiddenTrain, HiddenTrain);

vec eigval2 = eig_sym(C2);

double Max_lrCoeff2 = 1.0 / (2.0 * max(eigval2));

//###### maximum learnCoeff 2 value to be adopted #########

Declaring variables for the second part

New variables are initialized for the second part. Vector **dervPI2** (dimension *DimOut*) to store the values of the derivative of the information potential. Also, three matrices are initialized in this part of the model: **i)** the matrix to store the expanded particles obtained **OutputsTrain** (dimension *TrainDim* x *DimOut*), **ii)** the matrix to store the weight variation in each epoch **delta_weights2** (dimension *DimHidden* x *DimOut*), and **iii)** the matrix with the current weights of the second part of the autoencoder **Weights2** (dimension *DimHidden* x *DimOut*).

vector <double> dervPI2;

dervPI2.resize(DimOut);

mat OutputsTrain(TrainDim, DimOut);

mat delta_weights2(DimHidden,DimOut);

mat Weights2(DimHidden,DimOut);

Weights (2nd Half) initialization

The weights in the second part of the autoencoder may be initialized with the final weights obtained in the first part, using the transposed matrix, or with random numbers. When the variable **InitWeightsTranspose** is defined in the beginning of the model (see section D), the weights are initialized with the transposed matrix of the first part final weights, when **InitWeightsTranspose** is not defined, the weights are initialized with random numbers.

```
//the second part weights may be initialized with the transposed final weights from first part
#ifdef InitWeightsTranspose
   Weights2 = trans(Weights1);
// or with random numbers
#else
   for (unsigned int hid=0; hid <DimHidden; hid++)
   [
      for (unsigned int out=0; out<DimOut; out++)
      [Weights2(hid,out) = aleat(rng);]
   ]
#endif
```

Bias (2nd half) initialization

The bias values of the second half are always initialized with random numbers. Vector **Bias2** (dimension *DimOut*) store the bias of the second part.

```
//Random Initialization of vector Bias2
vec Bias2(DimOut);
for (unsigned int out=0; out<DimOut; out++)
[Bias2(out) = aleat(rng);]
```

Matrix OutputsTrain First Calculation

At this point, the weights2 and bias2 are initialized. The code proceeds with the calculus of the first matrix *OutputsTrain*, storing the initial outputs instances (using the initial weights2 and bias2).

```
//### Calculo inicial das particulas aumentadas ###
//faz o conjunto de particulas aumentado dado o conjunto de pesos actual, todas de uma vez
double somatorio = 0.0;
for (unsigned int tr=0; tr<TrainDim; tr++)
[
   for(unsigned int out=0; out<DimOut; out++)
   [
      for(unsigned int hid=0; hid<DimHidden; hid++)
      [
         somatorio += ((HiddenTrain(tr,hid))*(Weights2(hid,out)));
      ]
      somatorio += Bias2[out]; //VectorBias adicionado ao somatorio
      OutputsTrain(tr,out) = actFcn(somatorio, actFcnFlatness); //funçao activação
      somatorio = 0.0;
   ]
]
//### END Calculo inicial das particulas aumentadas ###
```

Variables initialization (2nd Half)

The code proceeds with the initialization of some variables that are further applied.

```
unsigned int Epochs2 = 0;
double MSE_train = 100000.0, Prev_MSE_train=0.0, MAE_train=0.0;
double Best_MAE_train=100.0, Best_MAE_test=100.0;
double Best_MSE_test=100.0, Best_MSE_train=100.0;
```

Epochs loop (2nd Half)

The code proceeds with the loop over the number of epochs to execute within the second half.

while (Epochs2 < numEpochs2)

[//(…)]

Traveling on instances

Once inside the loop over the number of epochs 2. The vector prevBias2 and the matrix prev_Weights2 store the values of the corresponding parameters concerning the previous epoch. The loop over the trainDim implements the classic backpropagation algorithm.

vec prevBias2(DimOut);

prevBias2 = Bias2;

mat prev_Weights2(DimHidden,DimOut);

prev_Weights2 = Weights2;

//##############################dPI/dzk ######################################

//derivada do potencial de informação em ordem a yi

for (unsigned int tr=0; tr<TrainDim; tr++)

[

for(unsigned int out=0; out<DimOut; out++)

[dervPI2[out] = -(TrainSet(tr,out) - OutputsTrain(tr,out));]

//nesta altura está calculada a derivada de PI em ordem a yi

//a seguir faz a derivada do yi em análise em ordem aos pesos e multiplica pela dPY/dyi e por rho, com o sinal de acordo com maximizar ou minimizar

for (unsigned int hid=0; hid<DimHidden; hid++)

[

for (unsigned int out=0; out<DimOut; out++)

[

//diferença das derivadas de y em ordem as pesos

delta_weights2(hid,out) = (-1.0)* lrCoeff2 * dervPI2[out] * (HiddenTrain(tr,hid) * DervActFcn(OutputsTrain(tr,out), actFcnFlatness));

```
        Weights2(hid,out) += delta_weights2(hid,out);
      ]
    ]

    //nota: a saida do espaço reduzido do neuronio bias é sempre 1
    for (unsigned int in=0; in<DimOut; in++)
    [Bias2(in) += (-1.0)* lrCoeff2 * (dervPI2[in]) * DervActFcn(OutputsTrain(tr,in), actFcnFlatness);]

    //fim da actualização dos pesos para o vector yi
    dervPI2.resize(DimIn);
]
```

With the new weights2 and Bias2 calculated within each epoch of backpropagation, the corresponding matrix OutputsTrain is calculated. This is followed by the saturation process.

MAE and MSE error calculation for the train data set

Each epoch performs the assessment of the error considering all particles. The error is measured with two metrics: the mean absolute error (MAE) and the mean square error (MSE). The MSE is used as the cost function to be minimized in the second half of the autoencoder, and consequently this measure is used to further define the *stop criterion* and the *learning coefficient 2 adaptive criterion*. Note that these error measures report the error obtained with the last epoch, since the particles are expanded at the beggining of each epoch. First the model starts to calculate the matrix *ErrorMatrixTrain* (dimension *TrainDim* x *DimOut*) which contains the difference between the Target (matrix *TrainSet*) and the Output (matrix *OutputsTrain*).

```
//################ Train data set MAE between target and output ##################
//Error matrix calculation
mat ErrorMatrixTrain (TrainDim, DimOut);
for(unsigned int tr=0; tr<TrainDim; tr++)
[
```

```
    for (unsigned int out=0; out<DimOut; out++)

    [ErrorMatrixTrain(tr,out) = TrainSet(tr,out) - OutputsTrain(tr,out);]

]
```

The MAE estimation is made calculating the sum of the absolute value of each element of *ErrorMatrixTrain*, and dividing this sum by the number of elements in the matrix *ErrorMatrixTrain*. The MAE value is stored in the variable ***MAE_train***.

```
//MAE calculation

MAE_train = 0.0;

for(unsigned int tr=0; tr<TrainDim; tr++)

[

    for(unsigned int out=0; out<DimOut; out++)

    [MAE_train += abs(ErrorMatrixTrain(tr,out));  ]

]

MAE_train = MAE_train/(dTrainDim*dDimOut);
```

The Best_MAE_train is intended to store the lowest MAE found over all epochs. This variable is compared with the MAE found for each epoch, and it is replaced whenever a lower MAE value is found.

```
//BEST condition

if (MAE_train<Best_MAE_train)[Best_MAE_train=MAE_train;]
```

The information on seed, Epochs2, MAE of the current epoch and Best MAE found is printed into the file *detailed_output*.

```
// Ficheiros detailed

detailed_output << "Seed; "<< seed <<";Epoca ;" << Epochs2 << "; MAE train ;"<< MAE_train << "; Best_MAE_train ;"<< Best_MAE_train;
```

The MSE is estimated with the sum of the square of all elements of the matrix *ErrorMatrixTrain*. This sum is then divided by the number of elements of this matrix.

```
//############ Train data set MSE between target and output #################

Prev_MSE_train = MSE_train;
```

```
MSE_train = 0.0;

for(unsigned int tr=0; tr<TrainDim; tr++)

[

   for(unsigned int out=0; out<DimOut; out++)

   [MSE_train += pow(ErrorMatrixTrain(tr,out),2.0);]

]

MSE_train = MSE_train/(dTrainDim*dDimOut);
```

The variable Best_MSE_train is intended to store the lowest MSE observed over all epochs. This variable is actualized whenever a lowest MSE_train is found.

```
//BEST condition

if (MSE_train<Best_MSE_train)[Best_MSE_train=MSE_train;]
```

The information concerning the MSE_train and the Best_MSE_Train is printed into the file *detailed_output*.

```
// Ficheiros detailed

detailed_output << "; MSE_Train ;" << MSE_train << "; Best_MSE_train ;" << Best_MSE_train;
```

Stop Criterion (2nd half)

The stop criteria in the 2nd half is similar to the one defined for the 1st half. The only difference is the value of the cost function used, this time it is used the minimization of the MSE.

```
//##### Stop Criterium #####

//if the MSE_treino is lower than the criterion goal, the epochs loop is closed

if ( abs( MSE_train) < goal)

[Epochs2 = numEpochs2;]

//##### End #####
```

Adaptive learning coefficient (2nd half)

The procedure followed with the adaptive learning coefficient in the 2nd half is similar to the one described for the 1st half of the autoencoder.

MAE and MSE error calculation for the test data set

The MAE and MSE are then calculated for the test data set. The code and the reasoning applied is similar to the one described for the train data set.

End of the Epochs2 loop

Finally, the number of epochs is incremented by one, till the while loop achieves the maximum number of epochs defined, or the stop criterion is met.

```
Epochs2++;
```

Time

Once the loops on learning coefficients and seed are closed, the model executes the calculation of the time (in seconds) spent, and includes this information in the last line of the results *csv file*. Finally, the results file is closed.

```
time (&end); //TIME

Resultados_LrCoeff << difftime (end,start) << "seconds" << endl; //TIME

Resultados_LrCoeff.close();
```

65 Bibliography

Beale, M., Hagan, M., & Demuth, H. (2012). MATLAB Neural Network Toolbox - User's Guide (R2012a).

El-Sharkawi, M. A. (1995). *Neural Network Application to High Performance Electric Drives Systems.* Paper presented at the Proceedings of the 1995 IEEE IECON 21st International Conference on Industrial Electronics, Control, and Instrumentation.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design*. Boston and London: Pws Pub.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation* (2nd ed.). Ontario, Canada: Pearson Education.

Jenssen, R., Principe, J., Erdogmus, D., & Eltoft, T. (2006). The Cauchy–Schwarz divergence and Parzen windowing: Connections to graph theory and Mercer kernels. *Journal of the Franklin Institute, 343*(6), 614-629.

Miranda, V. (2007). Redes Neuronais – Treino por Retropropagação (Texto de apoio à disciplina de Controlo Difuso e Redes Neuronais do 5º ano da LEEC). Porto, Portugal.

Principe, J. C. (2010). Information Theoretic Learning Renyi's Entropy and Kernel Perspectives: Springer.

Principe, J. C. (n.d.). Information-Theoretic Learning (Tutorial).

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. London, UK: Chapman & Hall/CRC.

Xu, D., & Principe, J. C. (1999). *Training MLPs Layer-by-Layer with the Information Potential.* Paper presented at the IJCNN International Joint Conference on Neural Networks, Washington.

Theoretical Concepts of BackPropagation Neural Networks

PTDC/EEA-EEL/104278/2008

Report LASCA / R6

66 1 Introduction

Training a neural network aims at optimizing a cost function F. For that optimization, we use the Steepest Descent Method (SDM) that is an iterative process that moves in direction to the optimum (it could be a local optimum):

$$\omega_{ij}{}^{k+1} = \omega_{ij}{}^{k} + \eta \frac{\partial F}{\partial \omega_{ij}} \qquad \text{eq. R6/(1)}$$

The steepest descent method basically will require adjustments in the weights that obey the following expression:

$$\Delta\omega_{ij} = \eta \frac{\partial F}{\partial \omega_{ij}} \qquad \text{eq. R6/(2)}$$

The parameter η corresponds to the iteration step, or learning rate, and it will be positive for maximization and negative for minimization. Generally, a momentum term is added to the algorithm. This term aims at avoiding the method to get trapped in local optima. Considering the momentum term, the algorithm can be defined as presented in (63). A detailed explanation can be found in (Miranda, 2007).

$$\Delta\omega_{ij}{}^{(t)} = \eta \frac{\partial F}{\partial \omega_{ij}} + \alpha \Delta\omega_{ij}{}^{(t-1)} \qquad \text{eq. R6/(3)}$$

67 2 Classic PROP theory overview

The classic backpropagation algorithm is a supervised approach to train neural networks, firstly proposed in (Rumelhart, Hinton, & Williams, 1986). The train of the neural network pursues the minimization of the

Mean Squared Error (MSE) between the output and target instances. The MSE is represented in eq. R6/(4), where N is the total number of instances and M the total number of output neurons. This expression is in agreement with the implementation followed at software MATLAB (Beale, Hagan, & Demuth, 2012).

$$MSE = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} (e_{ij})^2 = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} (T_{ij} - O_{ij})^2 \qquad \textbf{eq. R6/(4)}$$

And the derivative of MSE in order to the output O_{ij} is defined as follows.

$$\frac{\partial}{\partial O_{ij}} MSE = -\frac{2}{N \times M} (T_{ij} - O_{ij}) \qquad \textbf{eq. R6/(5)}$$

The minimization of MSE is made using a Steepest Descent Method (SDM) algorithm (Miranda, 2007). Moreover, the adaptive learning rate (Hagan, Demuth, & Beale, 1996, pp. 12-12) was applied to the classic PROP algorithm. The next figure details the autoencoder. As this is an autoencoder, output and target dimensions matches with input dimensions.

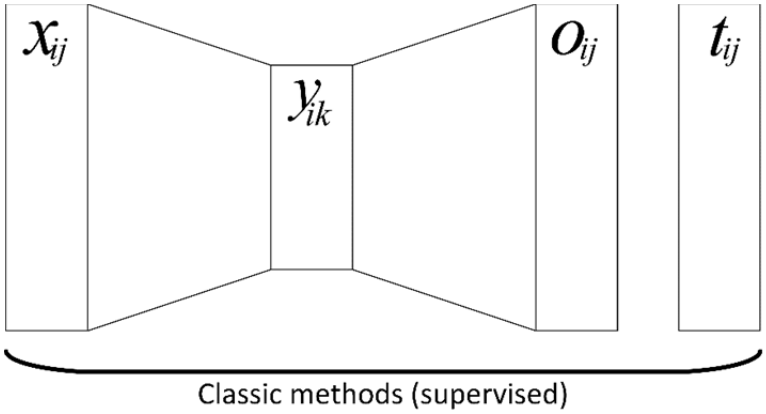


Figure R6/ 1 - Schematic of an autoencoder trained with backpropagation.

Where:

- x_{ij} = neuron at input layer;
- y_{ik} = neuron at hidden layer;
- o_{ij} = neuron at output layer;
- $t_{ij} = x_{ij}$ target neuron;
- $j = 1, \dots, M$; $k = 1, \dots, L$; $i = 1, \dots, P$, with M being the total number of input/output neurons, L the total number of hidden neurons, and P the total number of instances (examples) of training.

68 3 Other Theory Overview

68.1 3.1 MinMax Normalization Method

In training neural networks, should be done a pre-processing of data that consists in their normalization. The advantage of this procedure is that it reduces the effects of outliers in the data. The data have a better adjustment to the range of the activation function. As we use the tangent hyperbolic for the activation function, the range to our data is $[-1, 1]$. To perform this normalization we chose the Min-Max method:

$$data_{normalized} = \frac{data - min_{Orig}}{max_{Orig} - min_{Orig}}(max_{New} - min_{New}) + min_{New} \quad \textbf{eq. R6/(6)}$$

Where max_{New} is the maximum value for the normalized data; min_{New} is the minimum value for the normalized data; max_{Orig} is the maximum value for the original data; min_{Orig} is the minimum value for the original data. In our case, $max_{New} = 1$ and $min_{New} = -1$:

$$data_{normalized} = \frac{2 * data - max_{Orig} - min_{Orig}}{max_{Orig} - min_{Orig}} \quad \textbf{eq. R6/(7)}$$

There are other methods for data normalization but this method has the advantage of preserving exactly all relationships in the data. Another point in normalization procedure is the choose of max_{Orig} and min_{Orig}. In our code we implemented the methods:

- Global normalization: the chosen of max_{Orig} and min_{Orig} parameters can be done in all set of data (if they are similar, as our case);
- Normalization by entry: the chosen can be done by input matrix entry, this means that max_{Orig} and min_{Orig} choice need to be done for each input neuron in the correspondent set of instances.

Auxiliary calculations: Aim of normalizing data process is transform the data in the original range $[min_{Orig}, max_{Orig}]$ into a new range $[min_{New}, max_{New}]$.

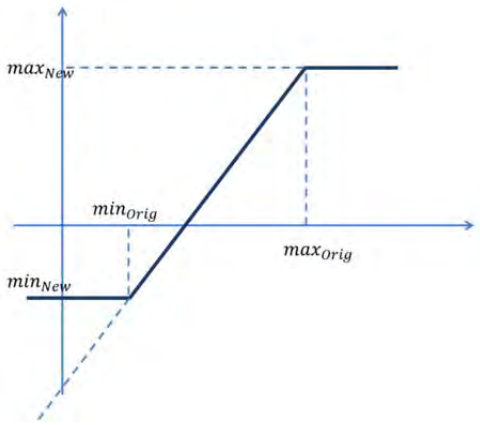


Figure R6/ 2 - Schematic of data transformation incurred with min max normalization.

We need the straight line equation: $y = m \cdot x + b$.

$$m = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}}$$

$$y = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot x + b$$

$(max_{Orig}, max_{New}) \in$ straight line, so:

$$max_{New} = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot max_{Orig} + b \Leftrightarrow$$

$$\Leftrightarrow b = max_{New} - \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot max_{Orig}$$

$$b = \frac{max_{New} \cdot max_{Orig} - max_{New} \cdot min_{Orig} - max_{New} \cdot max_{Orig} + min_{New} \cdot max_{Orig}}{max_{Orig} - min_{Orig}}$$

$$b = -\frac{max_{New} \cdot min_{Orig} - min_{New} \cdot max_{Orig}}{max_{Orig} - min_{Orig}}$$

So,

$$y = \frac{max_{New} - min_{New}}{max_{Orig} - min_{Orig}} \cdot x - \frac{max_{New} \cdot min_{Orig} - min_{New} \cdot max_{Orig}}{max_{Orig} - min_{Orig}}$$

$$\Leftrightarrow y = \frac{1}{max_{Orig} - min_{Orig}} \left[(max_{New} - min_{New}) \cdot x - (max_{New} \cdot min_{Orig} - min_{New} \cdot min_{Orig} + min_{New} \cdot min_{Orig} - min_{New} \cdot max_{Orig}) \right]$$

$$\Leftrightarrow y = \frac{1}{max_{Orig} - min_{Orig}} \left[(max_{New} - min_{New}) \cdot x - (max_{New} - min_{New}) \cdot min_{Orig} - min_{New} \cdot (min_{Orig} - max_{Orig}) \right]$$

$$\Leftrightarrow y = \frac{x - min_{\ Orig}}{max_{\ Orig} - min_{\ Orig}}(max_{\ New} - min_{\ New}) + min_{\ New}$$

This is,

$$data_{normalized} = \frac{data - min_{Orig}}{max_{Orig} - min_{Orig}}(max_{New} - min_{New}) + min_{New}$$

68.2 3.2 Adaptive Learning Rate

An adaptive learning rate was implemented in all autoencoders. The rules used to define the adaptive learning evolution are the ones define in (Hagan et al., 1996, pp. 12-12). These rules are detailed here. This section describes the influence of the parameters *learning rate* (***lr***), *learning rate increase* (***lr_inc***), *learning rate decrease* (***lr_dec***) and *maximum performance increase* (***max_perf_inc***) (we use the same name adopted in MATLAB neural networks toolbox for these parameters). The changes to induce in the learning rate are assessed by the value of the ratio between the cost function in the current epoch and the cost function obtained in the previous epoch:

$$Ratio = \frac{F_t}{F_{t-1}} \qquad \textbf{eq. R6/(8)}$$

The changes induced into the autoencoder parameters, considering different values of this ratio and depending on the minimization / maximization objective function, may include **i)** changes in learning rate, **ii)** acceptance or rejection of the changes in weights and bias of the current epoch, **iii)** changes in the momentum term rate. So far, the model does not include changes on the momentum term rate (note that the momentum term is set to zero in all the results further presented).

Minimizing case

When minimizing the cost function, the adaptive learning algorithm evolves differently under three distinct scenarios. The next table summarizes the three scenarios.

Table R6/ 1 - Different cases of adaptive learning algorithm evolution when minimizing the cost function.

Ratio value	Assessment	Learning Rate	Weights and Bias
]-∞,1[	The cost function evolves as desired.	The learning coefficient is increased by *lr_inc* rate (in our case always 1.01 –	The changes performed in the current epoch are maintained (weights, bias)

	The cost function is evolving in the desired direction (minimizing).	augmenting 1%).	
[1,max_perf_inc]	The cost function evolves in the opposite direction. "not severe" The cost function had evolved in the opposite direction to the one that was desired, but only 4%	The learning rate is not changed.	The changes of current epoch are maintained (weights and bias).
]max_perf_inc,+∞[	The cost function evolves in the opposite direction. "too severe" The cost function evolved drastically in the opposite direction to the one expected (more than 4%)	The learning rate is cut by *lr_dec* (in our case always 0.5).	The changes performed in the current epoch are discarded (weights and bias).

The first scenario occurs when the Ratio value is lower than 1, meaning that the cost function decreased. In this case, the algorithm accepts the changes performed in the structure of the autoencoder (weights and bias upgrades), and the learning coefficient is increased by *lr_inc*. The second scenario ("not severe") occurs for Ratio values comprised between [1, *max_perf_inc*]. Besides the cost function evolution being opposite

to the objective, the algorithm still accepts the changes made in this epoch and do not change the *lr*. The third scenario ("too severe") is considered for Ratio values higher than *max_perf_inc*. In this case the algorithm will discard the changes developed in the current epoch (this means that every change made under the current epoch is reverted, and the next epoch will have the same starting point). In this case the algorithm cuts the *lr* by *lr_dec*.

Maximizing case

The maximizing case (used in entropy of hidden layer and mutual information between inputs and hidden layer) is discussed in this section. Three possible situations may occur, as summarized in the next table.

Table R6/ 2 - Different cases of adaptive learning algorithm evolution when maximizing the cost function.

CostFn / PrevCostFn	Assessment	Learning Rate	Weights and Bias
]-∞, 1/max_perf_inc [	The cost function evolves in the opposite direction. "too severe" The cost function evolved drastically in the opposite direction to the one expected (more than 1/ *max_perf_inc*).	The learning rate is cut by *lr_dec* (in our case always 0.5).	The changes performed in the current epoch are discarded (weights and bias).
[1/max_perf_inc, 1]	The cost function evolves in the opposite direction. "not severe"	The learning rate is not changed.	The changes of current epoch are maintained (weights and bias).

	The cost function had evolved in the opposite direction to the one that was desired, but only till 1/ *max_perf_inc*.		
] 1,+∞[	The cost function evolves as desired. The cost function is evolving in the desired direction (maximizing).	The learning coefficient is increased by the *lr_inc* rate (in our case always 1.01 – augmenting 1%).	The changes performed in the current epoch are maintained (weights, bias)

The reasoning for these three situations is similar to the one previous described. The only difference is the use of the inverse value of *max_perf_inc* to define the limit between "not severe" and "severe" scenarios.

68.3 3.3 Stop Criteria

The stopping criterion is the number of epochs for all methods. When under a minimization objective, an extra stop criterion is defined, that will make the epochs loop stop for cost function values close to zero.

68.4 3.4 Neural Network Saturation

The saturation problem is appears when the nonlinear activation functions reach its upper or lower saturation limits. As El-Sharkawi explains (El-Sharkawi, 1995), any wide change in the input would produce no or minimal change in the output and the neurons in this case are paralyzed. El-Sharkawi also confirms that it is common and acceptable to have some neurons in the saturation region, but too many would render the neural network useless.

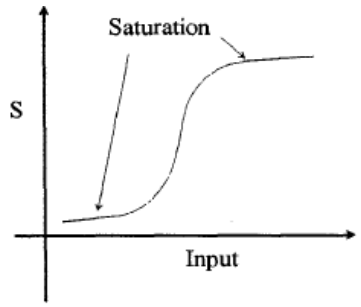


Figure R6/ 3 - Saturation regions of a Sigmoidal Function (from (El-Sharkawi, 1995)).

If network reaches saturation, the neurons must be randomly perturbed and the learning process continued. For El-Sharkawi, it is imperative that this saturation check be incorporated in any NN software.

69 4 PROP pseudo-code

```
Initialize[int numEpochs1; int DimIn; int DimHidden; int TestDim; int TrainDim; double momRate; double
actFcnFlatness; vec learnCoeff1; vec seed;]
Input [mat TrainDataSet]
Normalize [TrainDataSet]
Randomize [TrainDataSet]
Input [mat TestDataSet]
Normalize [TestDataSet]
Loop over learnCoeff1
[
  Loop over seed
  [
    Initialize [mat Weights1; mat Weights2; vec Bias1; vec Bias2;]
    Calculate [mat HiddenTrain; mat OutputsTrain; double max_learnCoeff1]
    Loop over Epochs1
    [
      Loop over TrainDim
      [
        Loop over DimIn
        [
          Calculate [dE/dy2;]
          Loop over DimHidden
          [
            Calculate [momTerm; dy2/dw; mat delta_weights2;]
            Update [Weights2;]
          ]
          Calculate [vec delta_Bias2;]
          Update [Bias2;]
        ]
        Loop over DimHidden
        [
          Loop over DimIn
          [
            Calculate[dE/dy2; dy2/dnet2; dnet2/dy1;]
          ]
          Calculate [dy1/dnet1;]
          Loop over DimIn
          [
            Calculate [dnet1/dw; momTerm; mat delta_weights1;]
            Update [Weights1;]
          ]
          Calculate [vec delta_Bias1;]
          Update [Bias1;]
        ]
      ]
      Update [HiddenTrain; OutputsTrain;]
      Calculate [mat ErrorMatrixTrain; MSE_train;]
```

```
    Verify [Stop criteria; Adaptive learnCoeff1 conditions]
    Calculate [MAE_train;]
    Calculate [mat HiddenTest; mat OutputsTest;]
    Calculate [mat ErrorMatrixTest; MSE_test; MAE_test;]
   ]
   Print [Results;]
  ]
]
```

70 5 PROP code documentation

This section details the code developed to train autoencoders with classic back propagation with learning adaptive criteria, according to the theory presented before.

Auxiliary Functions

actFcn

This function implements the choice for the activation function, with **activation** and **actFcnFlatness** as input parameters. We can choice between 3 functions: Logistic, Hyperbolic Tangent and Linear.

$$Logistic = \frac{1}{1 + e^{-\frac{\mathbf{activation}}{\mathbf{actFcnFlatness}}}}$$

$$Hyperbolic\ Tangent = \frac{2}{1 + e^{-\frac{\mathbf{activation}}{\mathbf{actFcnFlatness}}}} - 1$$

At MATLAB, $Hyperbolic\ Tangent = \frac{2}{1+e^{-2*\mathbf{activation}}} - 1$, that is equivalent to considering $\mathbf{actFcnFlatness} = {}^{1}/_{2}$.

$$Linear = \mathbf{activation}$$

To our runs, $Hyperbolic\ Tangent$ for MATLAB is the active activation function. The result of this function is a variable of type double, **output**.

DervActFcn

This function implements the derivative for the chosen activation function, with **output** and **actFcnFlatness** as input parameters.

$$Derivative\ Logistic = \frac{(\mathbf{output} * (1 - \mathbf{output}))}{\mathbf{actFcnFlatness}}$$

$$Derivative\ Tangent = \frac{\left(1 - \mathbf{output}^{2}\right)}{\mathbf{2} * \mathbf{actFcnFlatness}}$$

At MATLAB, $Derivative\ Tangent = \left(1 - \mathbf{output}^{\mathbf{2}}\right)$, that is equivalent to considering $\mathbf{actFcnFlatness} = {}^{1}/_{2}$.

$$Derivative\ Linear = 1$$

To our runs, $Derivative\ Hyperbolic\ Tangent$ for MATLAB is chosen according to the choice made for the activation function. The result of this function is a variable of type double, **tmpDerivative**.

PCA

This function implements the intrinsic function **princomp** of Library *Armadillo* for C++. The input parameters are **TrainSet_PCA** and *DimIn*. The output of *Armadillo* function **princomp** is a matrix (*DimInx DimIn*) of coefficients of principal components (*DimIn*).

The result of this function is a variable of type matrix, **Weights_PCA**.

//Auxiliar Functions

```
double  actFcn(double activation, double actFcnFlatness)
  [
    ////NEURON_ACT_LOGISTIC: // logistic activation function
    //double output = 1 / ( 1 + exp( - activation / actFcnFlatness ) );
    //NEURON_ACT_TANH:     // hyperbolic tangent (tanh)
    // double output = tanh( activation / actFcnFlatness );
    //double output = ( 2.0 / ( 1.0 + exp( - activation / actFcnFlatness ) ) ) - 1.0; //KEKO
    double output = ( 2.0 / ( 1.0 + exp( - 2.0* activation ) ) ) - 1.0; //matlab
    //double u = activation / actFcnFlatness;
    //double output = (exp(u) - exp(-u))/(exp(u)+exp(-u));
    ////NEURON_ACT_LINEAR:   // linear (ignores flatness)
    //double output = activation;
    return output;
  ]
```

```
double  DervActFcn(double output, double actFcnFlatness)
   [
      double tmpDerivative;
      ////NEURON_ACT_LOGISTIC: // logistic
      //tmpDerivative = ( output * ( 1 - output ) ) / actFcnFlatness;
      //NEURON_ACT_TANH: // hyperbolic tangent
      //tmpDerivative = ( 1.0 - pow( output , 2.0 ) ) / ( 2.0 * actFcnFlatness ); //keko
      tmpDerivative = ( 1.0 - pow( output , 2.0 ) ); //matlab

      ////NEURON_ACT_LINEAR: // linear
      //tmpDerivative = 1;
      return tmpDerivative;
   ]

mat PCA(mat TrainSet_PCA, unsigned int DimIn)
[
   mat Weights_PCA(DimIn,DimIn);
   Weights_PCA = princomp(TrainSet_PCA);
   return Weights_PCA;
]
```

Auxiliary parameters

The backpropagation code developed included similar specifications to those found within MATLAB. This goal was pursued by implementing the procedures as described in (Salman, 2008), which are the ones adopted in MATLAB libraries, alongside with the standard parameters' values as detailed in (Beale et al., 2012).

The **lr_inc** parameter is the Ratio to increase learning rate. When ratio between Cost Function actual value and Cost Function previous value is higher than one (i.e., when Cost Function increase) then learning rate is raised by a percentage of actual value. This percentage corresponds to the **lr_inc** value, in that case is assumed 5% of this actual value.

The **lr_dec** parameter is the Ratio to decrease learning rate. When ratio between Cost Function actual value and Cost Function previous value is less than a predetermined value (i.e., when Cost Function decrease) then learning rate is cut by a percentage of actual value. This percentage corresponds to the **lr_dec** value, in that case is assumed half of this actual value.

The predetermined value above is related with **max_perf_inc** parameter that is the Maximum performance increase. This is an acceptable interval to the increase of Cost Function, in our case 4%.

```
//###defining a vector containning the learning coefficients to test the redutor###

vector <double> learnCoeff;

double lr_dec = 0.5;

double lr_inc = 1.05;

double max_perf_inc = 1.04; //performance - racio (=CostFunction/Prev_CostFunction) a partir do qual nao se actualizam pesos

double goal = 0.00000001; //como no matlab
```

Vector learnCoeff

The code is intended to run several simulations, each one with a different learning coefficient. This is achieved with the definition of a vector **learnCoeff**, which includes all the desired values to test the learning coefficient. The construction of this vector can be done with a loop (when the values differ on the same slope) or with the individual inputs (when the values to test are distinct).

```
learnCoeff.push_back(0.005);

learnCoeff.push_back(0.05);

learnCoeff.push_back(0.5);

learnCoeff.push_back(5);


/*for (double f=0.001; f<0.011; f+=0.001)
```

```
[
    learnCoeff2.push_back(f);
]*/


unsigned int DimLC;
DimLC = learnCoeff.size();
//###end of defining a vector containning the learning coefficients to test the redutor###
```

The **value "DimLC"** measures the dimension of the vector containing the learning coefficients to test this method. This variable is further applied to define one of the main loops.

Main Results File

The code creates a *csv file*, with a specific name for Prop method – **Results_Prop.csv**, to further include the main information of tested autoencoder. This file will include information on the **i)** starting value of the learning coefficient used, **ii)** the final value of this coefficient (it evolve following an adaptive algorithm, which decreases it value whenever the respective cost function evolves to the opposite direction to the optimization objective), **iii)** the boost random seed used, **iv)** the best mean absolute error – **Best_MAE** and **v)** the best mean squared error – **Best MSE** between outputs and inputs of the autoencoder.

```
ofstream Resultados_LrCoeff;//ficheiros de saida de dados
string name ("Results_Prop.csv");
Resultados_LrCoeff.open(name);
```

Fixed Parameters

The main parameters are defined together. The variable **numEpochs1** define the number of epochs that the code will execute the autoencoder. **DimIn** is the number of neurons at the entrance of the autoencoder. **DimHidden** is the number of neurons in the middle of the autoencoder. **DimOut** is the exit number of neurons of the autoencoder and for this reason is assumed to be equal to the entrance number of neurons. The variables **dDimIn**, **dDimHidden** and **dDimOut** are doubles with the same meaning (the joint use of a variable for integer and double definition is not a good idea for some functions - therefore we never mix integers with doubles). The variable **TestDim** refers to the number of instances to be included in the validation data set. **dTestDim** is the double of **TestDim**. **TrainDim** is the number of instances to train the autoencoder. **dTrainDim** is the double of **TrainDim**.

The code includes the momentum rate (Miranda, 2007). The rate (α) is specified in the variable **momRate**. The variable **actFcnFlatness** refers to the activation function flatness (Haykin, 1999). The variables **SatLimit** and **ThresholdSat** are used in the Saturation procedure. Next, **Normalization** variable allows chooses if the input data normalization is processed or not. The variable **WeightsInitialization_PCA** should be defined when the weights initialization intended to be with Principal Component Analysis (PCA). When not defined the weights are initialized randomly with uniform distribution.

```
unsigned int numEpochs1 = 2000;


unsigned int DimIn = 24; // instances DimInension = first layer DimInension

double dDimIn = 24.0;


unsigned int DimHidden=16; //hidden layer dimension

double dDimHidden=16.0;


unsigned int DimOut = DimIn; //output layer dimension

double dDimOut = dDimIn;


unsigned int TestDim = 500;

double dTestDim = 500.0;

unsigned int TrainDim = 1000; // Numbers of instances/examples in train set

double dTrainDim = 1000.0;


double momRate = 0.0; //momentum coef

double actFcnFlatness = 0.5; // For Activation Function
```

Random numbers generators

The code uses the boost libraries (see www.**boost**.org) to define two functions that generate random numbers. The function **aux_aleat(rng)** generates uniform real numbers between the interval [0,1], the function **aleat(rng)** generates uniform real numbers between the interval [-1,1].

```
//#### random functions definition ####

boost::random::mt19937 rng;

boost::random::uniform_real_distribution<> aux_aleat(0,1);

boost::random::uniform_real_distribution<> aleat(-1,1);

//#### END of random functions definition ####
```

Randomizing the instances in train data set

The train data is a set of instances, or particles (each instance has the same size as the first layer of the autoencoder). The order in which the particles are covered is defined randomly (but the same order is used in every comparison of different methods, as the random seeds considered are the same: random seed = 3578). To create this random instance selection, the vector **indices** is created, which contains different integer numbers between 0 and *TrainDim*, and is further used to address the instances position on the train set.

```
// ##### Randomize TrainSet ######

rng.seed(3578);

vec RandPosition(TrainDim);

for (unsigned int i=0; i<TrainDim;i++)

[RandPosition(i) = aux_aleat(rng);]

uvec indices = sort_index(RandPosition);

// ##### END Randomize TrainSet ######
```

This process only interests for incremental learning strategy that updates weights after each instance is presented. The batch-mode learning strategy (first average the learning rule over all intances before changing weights) does not depend on way which intances are presented. Our code implements the batch-mode learning strategy.

Reading data from a csv file

The code reads two *csv files*: the train data set (`NovoTipoAC_1.csv`) and the validation data set (`NovoTipoAC_2.csv`). The code used to read both files is similar. This way, a provisional matrix is created (**ordered_TrainSet**) which contains all train instances organized in the same order as the input file. Next step is the data normalization procedure using the data contained in **ordered_TrainSet** recording the result in the normalized matrix **norm_ordered_TrainSet** (a description of this block can be seen in next section). A new matrix is created, the **TrainSet** (the one going to be applied in the further calculations). This matrix reads the information of the **norm_ordered_TrainSet** matrix using the **indices** vector to reorganize the instances.

Analogously, the matrix **TestSet** is created and corresponds to the validation set, which contains a set of instances to assess the performance of the trained autoencoder. There is no need to randomize the instances of the test data set, since they will not influence the train, and the error measured is not changeable with their order (this matrix is used only after the training is complete).

```
//##### Reading Training Data from csv file #######

ifstream indata;

double f;

string s="";

mat TrainSet(TrainDim,DimIn); // train set matrix

mat ordered_TrainSet(TrainDim,DimIn); // train set matrix

indata.open("NovoTipoAC_1.csv");

if(!indata) [ cerr << "Error: file could not be opened" << endl; exit(1);]

vector < double > auxiliar;

while(getline(indata, s, ';'))

[

   stringstream fs( s );

   fs >> f;

   auxiliar.push_back(f);

]
```

```
for(unsigned int k=0; k<TrainDim; k++)
[
   for (unsigned int j=0; j<DimIn; j++)
   [ordered_TrainSet(k,j) = auxiliar[(k*DimIn+j)];]
]
indata.close();
(DATA NORMALIZATION BLOCK – See next Section)

for(unsigned int k=0; k<TrainDim; k++)
[
   for (unsigned int j=0; j<DimIn; j++)
   [TrainSet(k,j) = norm_ordered_TrainSet(indices(k),j);]
]
ordered_TrainSet.clear();
norm_ordered_TrainSet.clear();
//##### END Reading Training Data from csv file #######
```

Data Normalization

The code allows to choose the data normalization procedure to adopt. Here it is important to know the meaning of data. Since the new range is $[-1, 1]$ (according to the activation function), the first step concerns the definition of the original range by entrance. Accordingly, **min_train** is the vector that contains the minimum of original data (*ordered_TrainSet*) by column and **max_train** is the vector that contains the maximum by column.

ordered_TrainSet = train data set from csv file (dim = *TrainDim x DimIn*)

min_train = minimum value by columns of data matrix *ordered_TrainSet* (dim = 1 x *DimIn*)

max_train = maximum value by columns of data matrix *ordered_TrainSet* (dim = 1 x *DimIn*)

The normalization by $\mathrm{column}(j)$ is done using the corresponding minimum and maximum value, **min_train(j)** and **max_train(j)**, according to eq. R6/(6) and (7).

```
// ##### Data normalization #####
#ifdef Normalization //performs the normalization by entrance
rowvec max_train = max(ordered_TrainSet,0);
rowvec min_train = min(ordered_TrainSet,0);

mat norm_ordered_TrainSet(TrainDim,DimIn);
for(unsigned int tr=0; tr<TrainDim; tr++)
[
   for (unsigned int in=0; in<DimIn; in++)
   [
      norm_ordered_TrainSet(tr,in) = (2*ordered_TrainSet(tr,in) - max_train(in) - min_train(in)) / (max_train(in) - min_train(in));
   ]
]
#endif
//##### End Data normalization #####
```

Similar code and description for test data set (*aux_TestSet*) normalization with correspondent minimum **min_test** and maximum **max_test**.

Learning Coefficient and seed loops

In Back Propagation method there are **two main loops**: for each value of **i)** learning coefficient and for each **ii)** seed. Since the learning coefficient was specified with a vector, the loop concerning learning coefficient relate to the position of **learnCoeff** vector to use. **Seed** loop was included to generalize results in what concern random initialization.

```
for (unsigned int v=0; v<DimLC; v++)
```

[

Open Output File for each LrCoeff

```
for(unsigned int seed=800; seed<850; seed+=5)
    [
        // #### Main Parameters Definition #####
        double lrCoeff1 = learnCoeff[v];
        // #### End of Main Parameters Definition #####
```

Next sub-sections: i) Weights initialization, etc.

```
    ] //end of seed loop
] //end of learning coefficient loop
```

The next sub-sections describe what is included within these two main loops.

The variable **DetailedOutputFile** when defined creates two *csv files*, containing the detailed information on the MAE and MSE errors observed in each epoch, one file relates to the training data set, and the second file concerns the test data set (note that the general results file includes the final MAE and MSE observed in each autoencoder).

Weights initialization

The weights of 1st half of the autoencoder can be initialized either with PCA values or with random numbers. These weights are initialized within the matrix **Weights1** of dimension of *DimIn x DimHidden*. **Weights2** is the matrix for the 2nd half of autoencoder and is inicialized using random numbers (dimension *DimHidden x DimOut*) or using the transpose of **Weights1**. Matrices **prev_Weights1** and **prev_Weights2** (dimension of *DimIn* x *DimHidden* and *DimHidden x DimOut*, respectively) of weights are defined to store in each epoch the weights of the previous epoch. These matrices are inialized to be a zero matrices.

PCA

PCA initialization is stored before loops related with learning coefficient and seed. Matrix **Coeff_PCA** store the coefficients of the principal components of train set, **TrainSet**, using PCA function defined before, that performs the Principal Components Analysis. In that method, **Weights1** stores, in its initialization, the coefficients of first *DimHidden* principal components. This is possible as $DimHidden \leq DimIn$. **Weights2** is defined as the transpose of **Weights1**.

```
// PCA
#ifdef WeightsInitialization_PCA
mat Coeff_PCA(DimIn,DimIn);
mat Weights1(DimIn, DimHidden);
mat Weights2(DimHidden, DimIn);
Coeff_PCA = PCA(TrainSet, DimIn);
Weights1 = Coeff_PCA.submat(span(), span(0, DimHidden-1));
Weights2 = trans(Weights1);
#endif
// END PCA
```

Random

Weights1 and **Weights2** matrices use the *aleat(rng)* function. These random numbers are repeatable for every experiment, as the seed used is defined in the second main loop (and this information is stored in the general results file as well).

```
//##### Weights Initialization #####
mat Weights1(DimIn,DimHidden), Weights2(DimHidden,DimOut);
rng.seed(seed);
for (unsigned int in=0; in<DimIn; in++)
[
   for (unsigned int hid=0; hid<DimHidden; hid++)
   [
      Weights1(in,hid) = aleat(rng);
      Weights2(hid,in) = aleat(rng);
   ]
]
//##### End of Weights Initialization #####
```

Bias initialization

In our code, bias are always initialized with random numbers, using the *aleat(rng)* function. Vectors **Bias1** (dimension *DimHidden*) and **Bias2** (dimension *DimOut*) stores the bias values for each epoch for the first and second part of autoencoder, respectively. Vectors **prevBias1** (dimension *DimHidden*) and **prevBias2** (dimension *DimOut*) stores the bias values of the previous epoch (initialized with zeros).

```
//##### Bias Initialization #####
vec Bias1(DimHidden), Bias2(DimOut);
vec prevBias1 (DimHidden), prevBias2(DimOut);
Bias1.zeros(); prevBias1.zeros(), Bias2.zeros(), prevBias2.zeros();
for (unsigned int hid=0; hid<DimHidden; hid++)
[Bias1(hid) = aleat(rng);]
for (unsigned int out=0; out<DimOut; out++)
[Bias2(out) = aleat(rng);]
//##### End of Bias Initialization #####
```

Calculating Hidden and Output matrices

Once the matrices *Weights* and the vectors *Bias* are initialized the model estimates the respective outputs (for the first and second part of the NN). This way, all train instances contained in *TrainSet* are transformed and stored in the matrix **HiddenTrain** (dimension *TrainDim* x *DimHidden*). Then, these instances in hidden layer are transformed and then stored in matrix **OutputsTrain** (*TrainDim* x *DimIn*). For each exit neuron, an induced local field is calculated, according to eq. R6/(9).

$$v_k = \sum_{in=1}^{DimIn} w_{k\,in} x_{in} + b_k\,, \qquad k = 1, \dots, DimHidden$$

Or

eq. R6/(9)

$$net_k = \sum_{in=1}^{DimIn} w_{k\,in} x_{in} + b_k\,, \qquad k = 1, \dots, DimHidden$$

In expression (9), the induced local field v_k for an exit neuron k is defined as the sum of all outputs from the previous layer (x_{in}) multiplied with the respective weights linked to neuron k ($w_{k\,in}$), and the bias weight

associated with the neuron k (b_k). For further theoretical information see (Haykin, 1999) on pages 34-36. This induced local field is then applied to the activation function previously defined, using the function **actFcn** with inputs v_k and actFcnFlatness. To compute the OutputsTrain matrix the formulas are similar, only changing the input variables:

$$v_k = \sum_{hid=1}^{DimHidden} w_{k\,hid} x_{hid} + b_k\,, \qquad k = 1, \dots, DimOut$$

Or

eq. R6/(10)

$$net_k = \sum_{hid=1}^{DimHidden} w_{k\,hid} x_{hid} + b_k\,, \qquad k = 1, \dots, DimOut$$

Where $w_{k\,hid}$ are the weights for the second part, x_{hid} are the elements of *HiddenSet* and b_k is the *Bias2*.

```
//##### Calculating HIDDEN and OUTPUT Matrices  #####
// HiddenTrain Matrix == FeedForward Step
mat HiddenTrain(TrainDim,DimHidden);  // HiddenTrain Matrix (dimensao padrões X tamanho hidden)
mat prev_HiddenTrain (TrainDim,DimHidden);
double aux1 = 0.0;
for (unsigned int tr=0; tr<TrainDim; tr++)
[
   for(unsigned int hid=0; hid<DimHidden; hid++)
   [
      for(unsigned int in=0; in<DimIn;in++)
      [
         aux1 += ((TrainSet(tr,in))*Weights1(in,hid));
      ]
      HiddenTrain(tr,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness);
      aux1 = 0.0;
   ]
```

```
]


prev_HiddenTrain = HiddenTrain;


// OutputsTrain Matrix == FeedForward Step
mat OutputsTrain(TrainDim,DimIn);  // OutputsTrain Matrix (dimensao padrões X tamanho hidden)
mat prev_OutputsTrain(TrainDim, DimIn);
double aux2 = 0.0;
for (unsigned int tr=0; tr<TrainDim; tr++)
[
   for(unsigned int in=0; in<DimIn; in++)
   [
      for(unsigned int hid=0; hid<DimHidden; hid++)
      [
         aux2 += ((HiddenTrain(tr,hid))*Weights2(hid,in));
      ]
      OutputsTrain(tr,in) = actFcn(aux2 + Bias2[in], actFcnFlatness);
      aux2 = 0.0;
   ]
]
prev_OutputsTrain=OutputsTrain;
//#####END of Calculating HIDDEN and OUTPUT Matrices  #####
```

Maximum Learning Coefficient Calculation

The learning coefficient should respect the threshold defined in eq. R6/(11), where η_1 is the learning coefficient for the first part of the NN, and λ_{MAX} is the maximum eigenvalue obtained from the variance and covariance matrix of the train data (Hagan et al., 1996).

$$\eta_1 \leq \frac{1}{2 \times \lambda_{MAX}} \qquad \text{eq. R6/(11)}$$

This threshold is calculated in the model and stored in the variable **Max_lrCoeff1,** see (Hagan et al., 1996, pp. 9-6).

```
mat C(DimIn,DimIn);

C = cov(TrainSet,TrainSet);

vec eigval = eig_sym(C);

double lambda = max(eigval);

double Max_lrCoeff1 = 1.0 / ( 2.0*lambda );
```

Cost functions variables

Before the *Epochs1 loop*, the code presents the inicialization of some variables used next: matrices **delta_weights1** and **delta_weights2** store the changes in weights in the actual epoch; **MAE_train** and **MSE_train** store the minimum absolute error and minimum square error for the train set, respectively (both initialized with zeros); the **Prev_CostFn** variable (initialized with zero) stores the cost function value of the previous epoch, and the **CostFn** variable (initialized with a big number) stores the cost function calculated in each epoch; **Best_MAE_train**, **Best_MAE_test**, **Best_MSE_train** and **Best_MSE_test** (set equal to 100 to be higher than *MAE_train* and *MSE_train* at first iteration). These variables are needed outside the *Epochs1 loop*, as they are further used to assess the stop criteria and to define the adaptive evolution of the learning coefficient.

```
mat delta_weights2(DimHidden,DimIn);

mat delta_weights1(DimIn,DimHidden);

double MAE_train = 0.0;

double MSE_train = 0.0;

double Prev_CostFn = 0.0, CostFn = 100000.0;//grande para criterio adapt do lrCoeff nao cortar na 1a iteração

double Best_MAE_train=100.0, Best_MAE_test=100.0;

double Best_MSE_test=100.0, Best_MSE_train=100.0;
```

Epochs loop

The epochs loop is declared as follows. The next sub-sections detail the processes that are included within this loop.

```
for (unsigned int Epochs1=0; Epochs1<numEpochs1; Epochs1++)
[
Next sub-sections(…)
]
```

Traveling over the instances

Here, the code implements the formulas corresponding to the Back Propagation method, according to the theory present at (Miranda, 2007). So, in this block, derivatives, gradient method, moment term, weights and bias update are computed, according to (Miranda, 2007). First for the second part of the autoencoder and then for the first part of autoencoder, as the name back Propagation suggests. "Back Forward Step" starts with loop *tr* presenting each instance to the network. For each instance (*tr*) and for each input neuron (*in*), the code computes the associated *sensitivity factor* (notation used at (Haykin, 1999, p. 185), in code defined as **error**. Then, for each neuron at hidden layer (*hid*), and consequently for each weights between input and hidden layer (*hid, in*), we have the declaration of moment term (**momterm**) and the update of the changes in weights (**delta_weights2**) and actual weights (**Weights2**). Vector bias is updated too (**Bias2**).

Similarly, sensitivity factor (**error**); derivatives (**D2** – derivative of activation function evaluated at output neurons (y_1), **DervInner** – partial derivative $\mathrm{d}E/\mathrm{d}y_2 * \mathrm{d}y_2/\mathrm{d}net_2 * \mathrm{d}net_2/\mathrm{d}y_1$, **D4** – derivative of activation function evaluated at hidden neurons); moment term (**momterm**); changes in weights (**delta_weights1**), weights (**Weights1**) and bias (**Bias1**) are updated for first half part of autoencoder, according to equations presented at (Miranda, 2007). The variables **prev_Weights1**, **prev_Bias1**, **prev_Weights2** and **prev_Bias2** store the corresponding values of the previous iteration and they are used to compute the momentum terms. "Back Forward Step" ends.

```
for (unsigned int tr=0; tr<TrainDim; tr++) //Cross samples: sample t
[
  // dV/Dw = dV/dy * dy/dw
  for(unsigned int in=0; in<DimIn; in++) //for each output neuron: derivative
```

```
    [
        //dV/dy
        double error = - ( 2.0/(dTrainDim*dDimIn) ) * (TrainSet(tr,in) - OutputsTrain(tr,in));

        //dy/dw = derv_ActFn_t * x_t
        for (unsigned int hid=0; hid<DimHidden; hid++)
        [
            double momterm = momRate*(Weights2(hid,in)- prev_Weights2(hid,in));//momentum term
            delta_weights2(hid,in) = (-1.0) * lrCoeff1 * (error * DervActFcn( OutputsTrain(tr,in), actFcnFlatness)
* HiddenTrain(tr,hid) ) + momterm;

            prev_Weights2(hid,in) = Weights2(hid,in);//momentum term
            Weights2(hid,in) += delta_weights2(hid,in);
        ]
        prevBias2(in) = Bias2(in);
        Bias2(in) += ((-1.0) * lrCoeff1 * error * DervActFcn(OutputsTrain(tr,in), actFcnFlatness)) + momRate *
(Bias2(in) - prevBias2(in));
    ]
    // dE/Dw = dE/dy2 * dy2/dnet2 * dnet2/dy1 * dy1/dnet1 *dnet1/dw
    double aux4 = 0.0;
    vector<double> DervInner;
    DervInner.clear();

    for(unsigned int hid=0; hid<DimHidden; hid++) //for each output neuron: derivative
    [
        for(unsigned int in=0; in<DimIn; in++)
```

```
    [
        //dE/dy2
        double error = - ( 2.0/(dTrainDim*dDimIn) ) * (TrainSet(tr,in) - OutputsTrain(tr,in));
        //dy2/dnet2
        double D2 = DervActFcn( OutputsTrain(tr,in), actFcnFlatness);
        //dnet2/dy1
        double D3 = Weights2(hid,in); aux4 += error * D2 * D3;
    ]
    DervInner.push_back(aux4); aux4 = 0.0;
    //dy1/dnet1
    double D4 = DervActFcn( HiddenTrain(tr,hid), actFcnFlatness);
    for(unsigned int in=0; in<DimIn; in++)
    [
        //dnet1/dw
        double D5 = TrainSet(tr,in);
        double momterm = momRate*(Weights1(in,hid)- prev_Weights1(in,hid));//momentum term
        delta_weights1(in,hid) =  (-1.0) * lrCoeff1 * ( DervInner[hid] * D4 * D5 ) + momterm ;
        prev_Weights1(in,hid) = Weights1(in,hid);
        Weights1(in,hid) += delta_weights1(in,hid);
    ]
    prevBias1(hid) = Bias1(hid);
    Bias1(hid) += (-1.0) * lrCoeff1 * (DervInner[hid]*D4) +  momRate * (Bias1(hid) - prevBias1(hid));
  ]
]
```

Hidden and Output matrices

When all instances (**TrainDim** instances) are "traveled", and the respective cumulated change on weights are complete (updated **Weights1** and **Weights2**), new output matrices are calculated, using the last changes performed on the weights (**Weights1**) (similarly to the first output matrices calculated).

```
// ##### Update HIDDEN and OUTPUT matrices #####
   // ##### HiddenTrain Matrix #####
   aux1 = 0.0;
   for (unsigned int tr=0; tr<TrainDim; tr++)
   [
      for(unsigned int hid=0; hid<DimHidden; hid++)
      [
         for(unsigned int in=0; in<DimIn; in++)
         [aux1 += ((TrainSet(tr,in))*Weights1(in,hid));]
         prev_HiddenTrain(tr,hid)=HiddenTrain(tr,hid);
         HiddenTrain(tr,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness); aux1 = 0.0;
      ]
   ]
Saturation Process (in next section)
   //####### OutputsTrain Matrix #####
   aux2 = 0.0;
   for (unsigned int tr=0; tr<TrainDim; tr++)
   [
      for(unsigned int in=0; in<DimIn; in++)
      [
         for(unsigned int hid=0; hid<DimHidden;hid++)
         [aux2 += ((HiddenTrain(tr,hid))*Weights2(hid,in));  ]
```

```
        prev_OutputsTrain(tr,in)=OutputsTrain(tr,in);

        OutputsTrain(tr,in) = actFcn(aux2 + Bias2[in], actFcnFlatness);

        aux2 = 0.0;

    ]

]
```

Saturation Process

One process to avoid the network saturation is randomly pertube the neurons. This was made by introducing a random perturbation in the weights. Next code implements this perturbation.

Definition of matrix **aux_saturation1** that identifies saturated elements of matrix *HiddenTrain* based on saturation limit (**SatLim**). If an element in *HiddenTrain* matrix is higher, in modulus, that the saturation limit then **aux_saturation1** stores in the corresponding place 1. For the non saturated elements stores 0. Definition of variable **aux_SatLevel1** that sums all the elements of matrix **aux_saturation1,** giving the total of its saturated elements. **SatLevel1** is the percentage of saturated elements. If the network is saturated, this is, if **SatLevel1** is higher than a threshold value (**ThresholdSat**) then weights will be pertubated according to:

$$W_{new} = (random * W) + W \qquad \textbf{eq. R6/(12)}$$

The $random$ number is defined as $0.1 * \text{aleat(rng)}$, where aleat(rng) generates uniform real numbers between the interval $[-1, 1]$, so $random$ is a uniform number between $[-0.1, 0.1]$. This means that weights suffer a perturbation of 10% into both directions.

```
//######## SATURATION process #####

mat aux_saturation1(TrainDim,DimHidden); //auxiliary matrix to calculate the proportion of weights within saturation state (the weights the absoulte value higher than 0.9)

aux_saturation1.zeros();


for (unsigned int tr=0; tr<TrainDim; tr++)

[

   for(unsigned int hid=0; hid<DimHidden;hid++)

   [
```

```
        if (abs(HiddenTrain(tr,hid))>=SatLimit)[aux_saturation1(tr,hid) =1;]
        else [aux_saturation1(tr,hid) =0;]
    ]
]
double aux_SatLevel1=accu(aux_saturation1); //retorna a soma acumulada de todos os elementos que estão contidos dentro da matrix auxiliar
double SatLevel1=aux_SatLevel1/(dTrainDim*dDimHidden); //retorna a percentagem de weights 1 que estão saturados

if(SatLevel1>ThresholdSat)
[
    for(unsigned int hid=0; hid<DimHidden;hid++)
    [
        for (unsigned int in=0; in<DimIn; in++)
        [Weights1(in,hid) =  0.1*aleat(rng)*Weights1(in,hid)+Weights1(in,hid);]
        Bias1(hid) = 0.1*aleat(rng)*Bias1(hid)+Bias1(hid);
    ]
    aux1 = 0.0;
    for (unsigned int tr=0; tr<TrainDim; tr++)
    [
        for(unsigned int hid=0; hid<DimHidden;hid++)
        [
            for(unsigned int in=0; in<DimIn;in++)
            [aux1 += ((TrainSet(tr,in))*Weights1(in,hid));]
            HiddenTrain(tr,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness);
            aux1 = 0.0;
```

]

]

]

//######## END SATURATION process #####

Cost function

The goal of this code is training a neural network minimizing the cost function, MSE.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(T_i - O_i)^2 \qquad \textbf{eq. R6/(13)}$$

Evaluating MSE at all neurons of all instances, the cost function becames:

$$CostFunction = \frac{1}{TrainDim \cdot DimIn}\sum_{tr=1}^{TrainDim}\sum_{in=1}^{DimIn}(T_{tr,in} - O_{tr,in})^2 \qquad \textbf{eq. R6/(14)}$$

In code, matrix **ErrorMatrixTrain** corresponds to $(T_{tr,in} - O_{tr,in})$, where T is the target (in that case, is the *TrainSet* matrix) and O is the output (in that case, is the *OutputsTrain* matrix). Defined **Prev_CostFn** is the value of cost function in previous epoch; **CostFn** (initialezd with zero) is the value of cost function in actual epoch.

```
mat ErrorMatrixTrain (TrainDim, DimIn);
for(unsigned int tr=0; tr<TrainDim; tr++)
[
   for (unsigned int in=0; in<DimIn; in++)
   [
      ErrorMatrixTrain(tr,in) = TrainSet(tr,in) - OutputsTrain(tr,in);
   ]
]
Prev_CostFn = CostFn; CostFn = 0.0;
for(unsigned int tr=0; tr<TrainDim; tr++)
```

```
[
   for(unsigned int in=0; in<DimIn; in++)
   [CostFn += pow(ErrorMatrixTrain(tr,in),2.0);]
]
CostFn = CostFn/(dTrainDim*dDimIn);
```

Stop criteria

At this point, the new value of the cost function is calculated (i.e. the MSE of *TrainSet* at the end of each epoch). The model includes a stop criterion, which is activated when the magnitude increment of the cost function is really low (in this case, when the value of the cost function in the current epoch is lower than **goal** parameter. Note that this criterion is only active when the model is minimizing the objective function.

```
//##### Stop Criterium #####
//if the cost function do not evolve more than the criteria, than the loop of the epochs is closed
if ( abs(CostFn) < goal)[Epochs1 = numEpochs1;]
```

Adaptive learning coefficient

The model includes the adaptive evolution of the learning coefficient. The adaptive criterion is activated when the cost function evolves in the opposite direction to that expected. The implementation of this evaluation is made with the comparison of cost function values obtained in the previous and current epochs (using the ratio between the current cost function and the cost function obtained in the previous epoch, which is compared with 1 and with the variable ***max_perf_inc***), and taking in consideration the maximization or minimization objective of the problem.

In Back Propagation algorithm, the goal is minimize MSE. In this case, it is expected the ratio ***CostFn/Prev_CostFn*** to be lower than 1, which means that the current epoch allowed the achievement of a lower value to the cost function. This way, when the ratio ***CostFn/Prev_CostFn*** is lower than 1, the model accepts the changes in weights and bias, and increases the learning rate (**lrCoeff1**) by **lr_inc**. For this case does not accept a value for learning rate higher than **Max_lrCoeff1**. When the ratio is higher than 1, two cases can occur: the “not severe” case or the “too severe” case, with similar rules to the ones exposed for the maximization case.

The “not severe” occurs when the ratio ***CostFn/Prev_CostFn*** is comprised between 1 and *max_perf_inc*. The “too severe” occurs when the ratio ***CostFn/Prev_CostFn*** is higher than *max_perf_inc*. In first case,

accepts weights/bias and do not change the learning rate. In second case, do not accept weights/bias (use the previous ones) and cuts the learning rate by **lr_dec**. This second case do not accept **lrCoeff1** lower than 0.00001.

```
//##### Learning Coeff Adaptive #####
if( CostFn/Prev_CostFn > max_perf_inc)
[
   lrCoeff1 =  lrCoeff1 * lr_dec; //corta o learning rate
   Weights1 = prev_Weights1; //não aceita pesos
   Weights2 = prev_Weights2;
   Bias1= prevBias1;
   Bias2= prevBias2;
   OutputsTrain=prev_OutputsTrain;
   HiddenTrain=prev_HiddenTrain;
   if (lrCoeff1 < 0.00001)[lrCoeff1 = 0.00001;]
]
if( CostFn/Prev_CostFn < 1.0)
[
   lrCoeff1 =  lrCoeff1 * lr_inc; //aceita pesos e aumenta learning rate
   if(lrCoeff1 > Max_lrCoeff1)[lrCoeff1 = Max_lrCoeff1;]
]
if((CostFn/Prev_CostFn >= 1.0) && (CostFn/Prev_CostFn <= max_perf_inc))
[
   //aceita os pesos
   //não altera o learning rate
]
//##### End Learning Coeff Adaptative #####
```

Train MAE and MSE Calculation

In this section, an evaluation of MAE (Mean Absolute Error) and MSE (Mean Square Error) of *TrainSet* is computed. First, compute the MAE (**MAE_train**) and the **Best_MAE_train** that is the best value (minimum) that MAE reaches until that epoch. Do similar to MSE for *TrainSet*.

```
MAE_train = 0.0;
for(unsigned int tr=0; tr<TrainDim; tr++)
[
   for(unsigned int in=0; in<DimIn; in++)
   [MAE_train += abs(ErrorMatrixTrain(tr,in));]
]
MAE_train = MAE_train/(dTrainDim*dDimIn);
if (MAE_train<Best_MAE_train)[Best_MAE_train=MAE_train;]
MSE_train=CostFn;
if (MSE_train<Best_MSE_train)[Best_MSE_train=MSE_train;]
```

Hidden and Output TEST Matrices

Similar to the ones for *TrainSet*, **HiddenTest** and **OutputsTest** matrices are calculated based on last updated weights and bias.

```
//compression
mat HiddenTest(TestDim, DimHidden);
aux1 = 0.0;

for (unsigned int te=0; te<TestDim; te++)
[
   for(unsigned int hid=0; hid<DimHidden; hid++)
   [
      for(unsigned int in=0; in<DimIn; in++)
      [aux1 += ((TestSet(te,in))*Weights1(in,hid));]
```

```
        HiddenTest(te,hid) = actFcn(aux1 + Bias1(hid), actFcnFlatness); aux1 = 0.0;
    ]
]
//expansion
mat OutputsTest(TestDim, DimIn);
aux2 = 0.0;

for (unsigned int te=0; te<TestDim; te++)
[
    for(unsigned int in=0; in<DimIn; in++)
    [
        for(unsigned int hid=0; hid<DimHidden; hid++)
        [aux2 += ((HiddenTest(te,hid))*Weights2(hid,in));]
        OutputsTest(te,in) = actFcn(aux2 + Bias2(in), actFcnFlatness); aux2 = 0.0;
    ]
]
```

Test MAE and MSE Calculation

Similar to the ones for *TrainSet*, **ErrorMatrixTest**, **MAE_test**, **Best_MAE_test**, **MSE_test** and **Best_MSE_test** are calculated using the *TestSet* matrix.

```
mat ErrorMatrixTest (TestDim, DimIn);
for(unsigned int te=0; te<TestDim; te++)
[
    for (unsigned int in=0; in<DimIn; in++)
    [ErrorMatrixTest(te,in) = TestSet(te,in) - OutputsTest(te,in);]
]
double MAE_test = 0.0;
```

```
for(unsigned int te=0; te<TestDim; te++)
[
   for(unsigned int in=0; in<DimIn; in++)
   [MAE_test += abs(ErrorMatrixTest(te,in));]
]
MAE_test = MAE_test/(dTestDim*dDimIn);

if (MAE_test<Best_MAE_test)[Best_MAE_test=MAE_test;]
double MSE_test = 0.0;
for(unsigned int te=0; te<TestDim; te++)
[
   for(unsigned int in=0; in<DimIn; in++)
   [MSE_test += pow(ErrorMatrixTest(te,in),2.0);]
]
MSE_test = MSE_test/(dTestDim*dDimIn);
if (MSE_test<Best_MSE_test)[Best_MSE_test=MSE_test;]
```

At this point the epochs loop is complete.

Results files

After the loop including all epochs is complete, the model is now ready to provide the final results: the initial value of learning coefficient 1, the final value of the learning coefficient 1 (the final value after running all *Epochs1*, which may include several cuts due to the adaptive procedure implemented), the seed for random numbers used in this experiment, and MAE and MSE values obtained in this experiment.

Time

Once the loops on learning coefficients and seed are closed, the model executes the calculation of the time (in seconds) spent, and includes this information in the last line of the results *csv file*. Finally, the results file is closed.

```
time (&end); //TIME
```

```
Resultados_LrCoeff << difftime (end,start) << "seconds" << endl; //TIME

Resultados_LrCoeff.close();
```

71 Bibliography

Beale, M., Hagan, M., & Demuth, H. (2012). MATLAB Neural Network Toolbox - User's Guide (R2012a).

El-Sharkawi, M. A. (1995). *Neural Network Application to High Performance Electric Drives Systems.* Paper presented at the Proceedings of the 1995 IEEE IECON 21st International Conference on Industrial Electronics, Control, and Instrumentation.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design*. Boston and London: Pws Pub.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation* (2nd ed.). Ontario, Canada: Pearson Education.

Miranda, V. (2007). Redes Neuronais – Treino por Retropropagação (Texto de apoio à disciplina de Controlo Difuso e Redes Neuronais do 5º ano da LEEC). Porto, Portugal.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by back-propagating errors. *Letters to Nature, 323*(9), 533-536.

Salman, M. (2008). Adaptive Learning Rate Versus Resilient BackPropagation for Numeral Recognition. *J. of alanbar university for pure science*.

Comparative Analysis between ITL and BackPropagation autoassociative neural networks in power system applications

PTDC/EEA-EEL/104278/2008

Report LASCA / R7

72 1 Introduction

This report summarizes the results obtained from the performance comparison of autoassociative neural networks trained with information criteria (Principe, 2010) and with classic backpropagation (Rumelhart, Hinton, & Williams, 1986). The theoretical description of information criteria implemented is detailed in report (Palma & Hora, 2012), and of backpropagation algorithm in report (Palma & Martins, 2012). A detailed description and statistical analysis of the data used is provided in the report (Hora & Palma, 2012).

This report is organized as follows. Section 2 describes main aspects concerning the data used and the respective normalization employed. Section 3 includes some test experiments conducted with the objective of validating the implementation made in C++. This validation is further confirmed with the similarity of results obtained from the C++ platform and the homologous experiment made in MATLAB. Finally, section 4 includes the results for two case studies, explored with the simulation of different scenarios. The anaylisis of the reported experiments suggests that the ITL networks achieve more accurate results than the classic backpropagation algorithm.

73 2 Data

Next results are related to data of wind velocity. Our total data is a set of ninety-six day types with 24 hours each day (96 types x 10000 days for each type). For this work we use type 1 and 3 to construct the train and test sets, as it is further detailed. Data was normalized to range between $[-1, 1]$ using the MinMax normalization method (Palma & Hora, 2012; Palma & Martins, 2013). For each experiment, the data set

applied within the train phase included 1000 examples, and the data set applied to the test phase included 500 examples, different from the ones used within the train set.

In order to statistically verify that the two data sets derive from the same distribution, the Kolmogorov test was applied, considering each hour separately. The base scenario uses data from the type 1 only. The train data set is composed of 1000 type 1 examples, and the test data set is composed of 500 type 1 examples, different from those used for train. According to (Hora & Palma, 2012), the assumption that both data sets come from the same distributions is sustained.

The multimodal scenario applies data from type 1 and from type 3, into equal halves. Again, the train data set has 1000 examples (500 of type 1 and 500 of type 3), and the test data set is composed of 500 examples (250 from type 1 and 250 from type 3). Using a similar reasoning as the one former described for base scenario, the Kolmogorov test was applied to infer on the similarity of distributions of the train and test data sets. Again, the assumption of distribution similarity over the two data sets is kept. The multimodal scenario has a second assumption concerning the data applied: the type 1 and type 3 data derive from different distribution. The Kolmogorov test was used to prove that the type 1 distribution is different from the type 3 distribution, and the dissimilarity of these two distributions was statistically significant.

74 3 Comparing BackPropagation in C++ with MATLAB

MATLAB Software provides the *Neural Network Toolbox* to simulate neural networks, which includes the Back Propagation (PROP) method. To validate the classic PROP code developed in C++, some runs in both approaches were compared. Table R7/ 1 summarizes the parameters adopted in all simulations compared. Table R7/ 2 shows the performance criteria (mean absolute error MAE and mean squared error MSE) for these runs, which were conducted under similar conditions.

Table R7/ 1 – Parameters adopted for the simulation of neural networks.

Network structure	24 – 12 – 24
Train Set	1000 examples
Epochs	1000
Random Seed	800 (same weights, bias and order in train set)
Activation Function	tanh (= tansig @Matlab)

Momentum Rate	0
Goal	10^{-8} (stop criterium parameter @Matlab)
Learning Coefficient	0.1 / 0.01 / 0.001
Lr_dec	0.5 / 0.7 (Learning Coefficient Adaptive parameter @Matlab)
Lr_inc	1.01 / 1.05 (Learning Coefficient Adaptive parameter @Matlab)
Max_perf	1.04 (Performance function parameter @Matlab)

Table R7/ 2 - Errors obtained with the PROP implemented in C++ and with the MATLAB Neural Network Toolbox, considering the same data, specifications and initialization.

			PROP @ C++		PROP @MATLAB	
lr_dec	lr_inc	lc	MAE	MSE	MAE	MSE
0.5	1.05	0.1	0.1823	0.0529	0.18597	0.05504
0.5	1.05	0.01	0.183	0.0533	0.18739	0.05583
0.5	1.05	0.001	0.185596	0.0548047	0.18917	0.05668
0.5	1.01	0.1	0.201862	0.0642166	0.19003	0.05738
0.5	1.01	0.01	0.204377	0.0658088	0.1961	0.06088
0.5	1.01	0.001	0.210361	0.0696838	0.2076	0.06789
0.7	1.05	0.1	0.182272	0.0529872	0.18663	0.05547
0.7	1.05	0.01	0.190464	0.0576137	0.18774	0.05616
0.7	1.05	0.001	0.18433	0.0541137	0.1902	0.05755
0.7	1.01	0.1	0.198079	0.0619335	0.19234	0.05864
0.7	1.01	0.01	0.203593	0.0653055	0.20051	0.06344

0.7	1.01	0.001	0.2106	0.0698393	0.20768	0.06792

In dark green cells are the best values for the programed runs that correspond to the same run in both codes. Similarly, second best run (light green) and worst run (red) correspond to the same runs too.

75 4 Exploring ITL networks with Scenarios

In order to conclude on the performance obtained with the ITL concepts applied to neural networks, a base scenario was defined, from where other scenarios were constructed (see Figure R7/1). Each new scenario includes a single change, so the corresponding feature may be assessed.

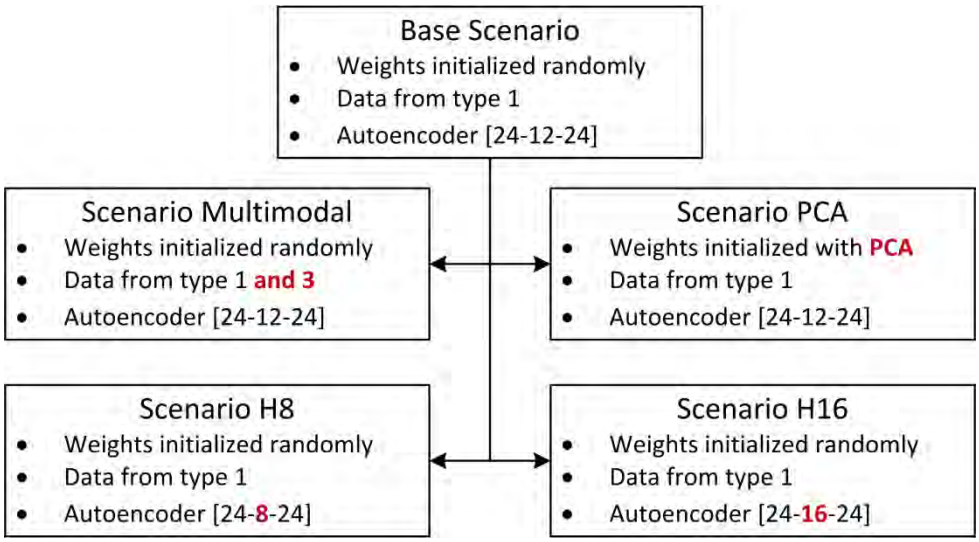


Figure R7/ 1 - Experimental Derived Scenarios from Base Scenario.

The parameters used in Base Scenario are listed next:

- **Train Set:** 1000 examples
- **Test Set:** 500 examples
- Number of ITL Seeds: 10
- Number of Prop Seeds: 10
- **Sigma:** calculated based on Silverman's Rule
- Number of ITL Epochs = 2000
- Number of Prop Epochs = 2000
- **ITL Learning Coefficient (1ª half), C1** $=\{0.005, 0.05, 0.5, 5\}$ (initial values)
- ITL Learning Coefficient (2ª half), C2 $=\{0.005, 0.05, 0.5, 5\}$ (initial values)
- Prop Learning Coefficient (Supervised), C = $\{0.005, 0.05, 0.5, 5\}$ (initial values)

Next table resumes the information specific of each experiment, taking the base scenario as a reference:

Table R7/ 3 - Experiments/Scenarios and its specifications.

Experiment		Architecture	Data	Synaptic Weights Inicialization
Code	Label			
T5	Normal Base Scenario	24 – 12 – 24	Normal	random
T6	Normal H16 Scenario	24 – 16 – 24	Normal	random
T7	Normal H8 Scenario	24 – 8 – 24	Normal	random
T8	Normal PCA Scenario	24 – 12 – 24	Normal	PCA
T9	Multi Base Scenario	24 – 12 – 24	Multi	random
T10	Multi H16 Scenario	24 – 8 – 24	Multi	random
T11	Multi H8 Scenario	24 – 16 – 24	Multi	random
T12	Multi PCA Scenario	24 – 12 – 24	Multi	PCA

Next, a big plan resuming the results obtained is presented for each above experiment. The results are split in two groups: unsupervised and supervised. In first results line of big plan, corresponding to the scenario label (eg, "Normal Base Scenario"), a graph showing the MSE Test versus Best MSE Train is presented for each ITL method and for BackPropagation method. The end of column presents a comparative of these four graphs for MSE Test. Second line refers to the iterations needed to converge in first part ("1st") and in second part ("2nd") of AA. The next line ("Error distribution"), presents a box plot, constructed for each method, concerning the 10 seeds runned for the best learning rate combination achieved for the experiment. "Cost Function" line presents the evolution of cost function (Cauchy-Schwartz Mutual Information, Euclidean Distance Mutual Information and Entropy, respectively). This topic is not available for Backpropagation. The table, in last column of this line, numerically resumes the MSE (train, test and

validation) results for the best seed. Experiments *T6*, *T7*, *T8*, *T10*, *T11*, *T12* have an additional line, where a comparison with base scenario is done, referring to the MSE Test.

The **criterion** to choose the best coefficients is the minimum MSE found on all simulations performed (10 for each combination of parameters) for each autoencoder.

76 5 Discussion and Conclusions

From experiment **T5** we conclude that MI_ED (MSE Test=0.0379) performs better than the other approachs. In fact, MI_ED, MI_CS and Entropy have similar results. The worst result was for the PROP method (MSE Test=0.0647).

Concerning experiment **T6**, the training criterion returning the best performance was was Entropy, with a MSE of 0.0255 referring to the test dataset. The second best criterion was the MI_ED, with a MSE Test of 0.0256. The criterion MI_CS returned a MSE Test value of 0.0298, followed by the PROP criterion, with a MSE Test value of 0.0621.

Similar analyses were performed for other experiments. The methods found to better perform are MI_ED and entropy, followed by MI_CS. The PROP algorithm performed worst than the ITL approachs for all experiments, with the exception of experiment T8, where the synaptic weights were initialized with PCA, leading PROP to quickly achieve high quality solutions.

Also, PROP and MI_ED were the methods associated with higher variability concerning the error distribution (out of 10 runs).

MI_CS provided significantly faster simulations than the remainder ITL methods. For that reason, and because the quality results were similar amont the all ITL approachs, that method was adopted to be applied in future research.

77 Bibliography

Hora, J., & Palma, V. (2012). Analysis using descriptive statistics on the data used for the ITL networks and on the data used in the Topology problem (Power System) *INESC Interim Reports*. Porto, Portugal: INESC TEC.

Palma, V., & Hora, J. (2012). Theoretical Concepts of ITL Neural Networks *INESC Interim Report*. Porto, Portugal.

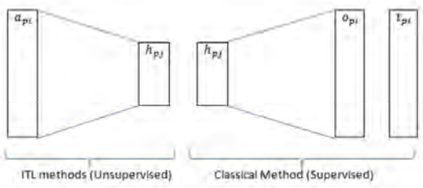
Palma, V., & Martins, J. H. (2012). Theoretical Concepts of BackPropagation Neural Networks *INESC Interim Report*. Porto, Portugal: INESC TEC.

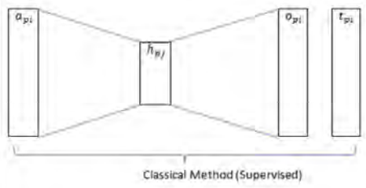
Palma, V., & Martins, J. H. (2013). Training Neural Networks - Theory of Practical Issues *INESC Interim Report*. Porto, Portugal: INESC TEC.

Principe, J. C. (2010). Information Theoretic Learning Renyi's Entropy and Kernel Perspectives: Springer.

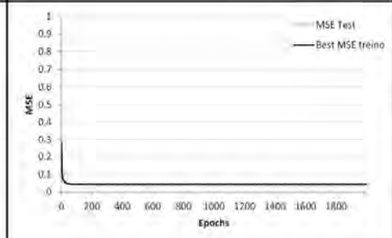
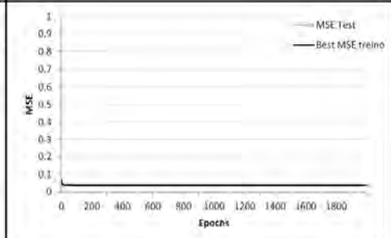
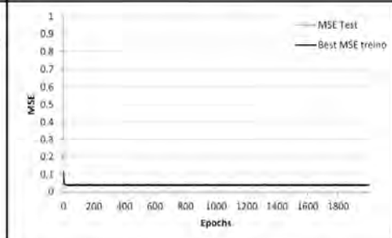
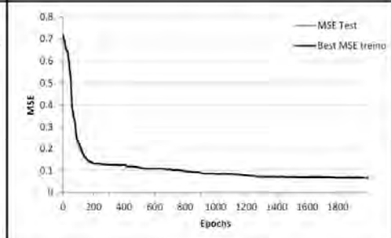
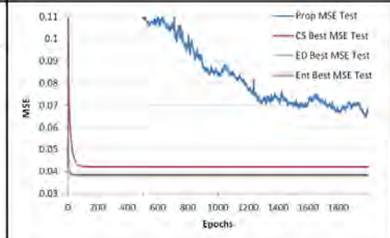
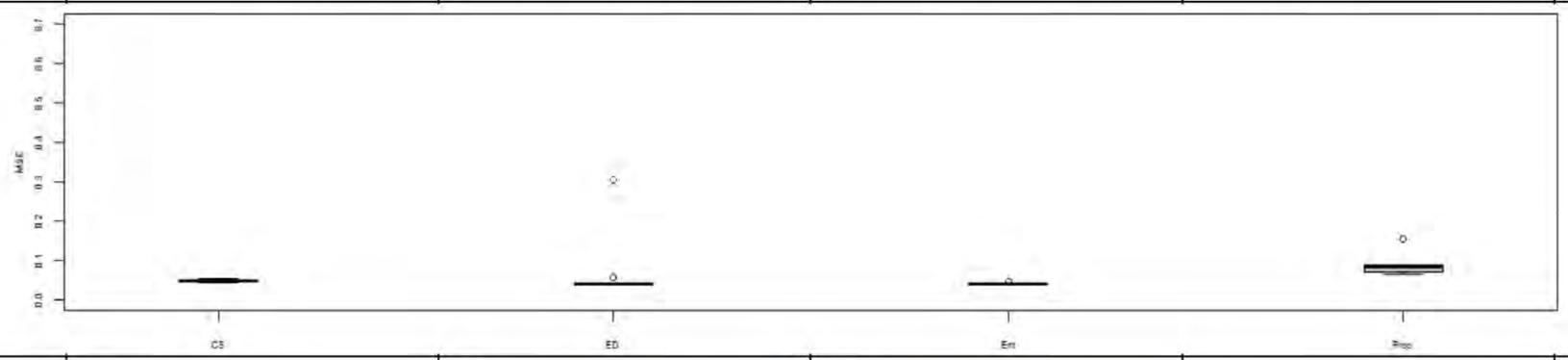
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by back-propagating errors. *Letters to Nature, 323*(9), 533-536.

Annex 1 - Experiment T5

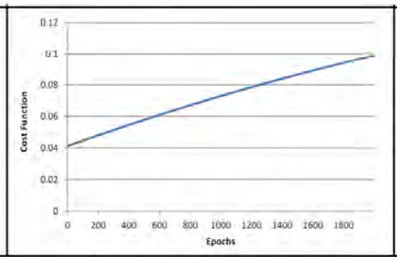
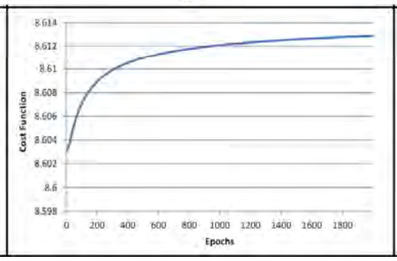
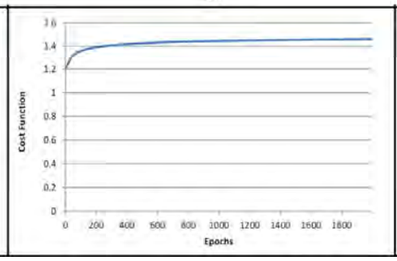
<table>
<tr><td rowspan="2">Scenario</td><td colspan="3">Unsupervised Learning
 

Iterations 1st # Iterations 2nd</td><td>Supervised Learning
 

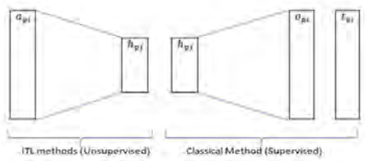
Iterations</td><td>Architecture: 24 – 12 – 24

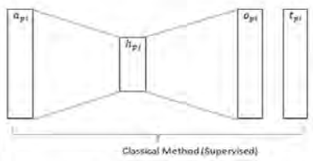
Data: Source 1 (Normal)
Normalization: By Entrance
InitializeWeights: random
- Best combination of parameters
- Best Seed</td></tr>
<tr><td>MI Cauchy Swartz</td><td>MI Euclidean Distance</td><td>Entropy</td><td>BackPropagation</td><td>Comparative</td></tr>
<tr><td>Normal Base Scenario</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>Description</td><td># Iterations: 2000 (1st) + 178 (2nd)</td><td># Iterations: 1988 (1st) + 79 (2nd)</td><td># Iterations: 2000 (1st) + 121 (2nd)</td><td># Iterations: 2000</td><td></td></tr>
<tr><td colspan="5"> 

Error Distribution

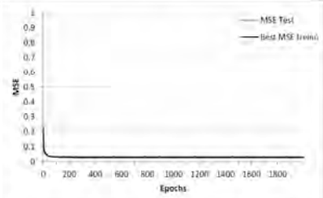
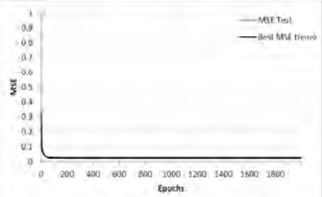
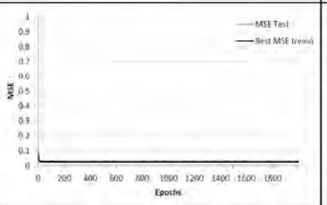
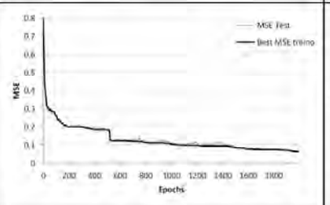
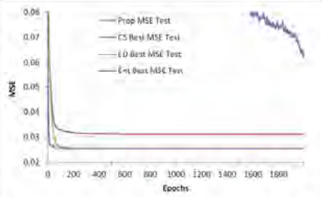
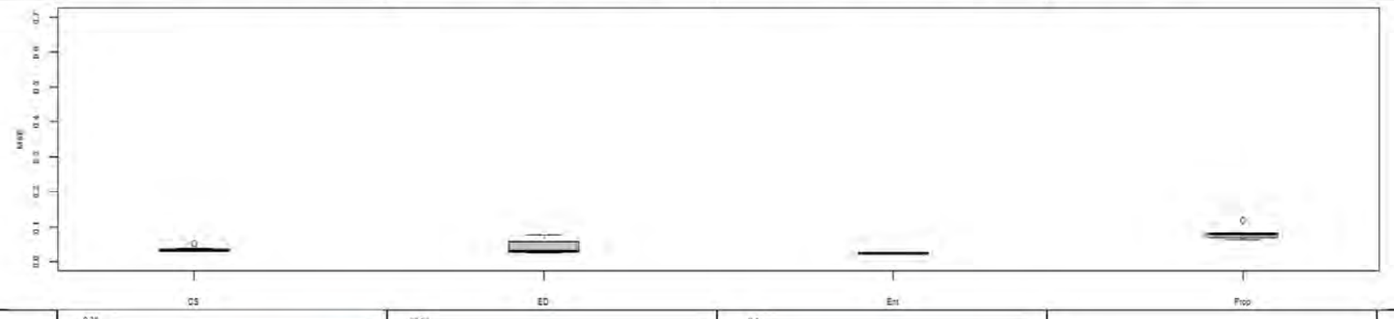
10 seeds for the Best Learning Rate Combination {c1,c2}</td></tr>
<tr><td>Cost Function (1st part)</td><td> 
</td><td> 
</td><td> 
</td><td>N.A.</td><td><table><tr><th>MSE</th><th>Train</th><th>Test</th><th>Validation</th></tr><tr><td>CS</td><td>0.0423</td><td>0.0421</td><td>0.0435</td></tr><tr><td>ED</td><td>0.0383</td><td>0.0379</td><td>0.0393</td></tr><tr><td>Ent</td><td>0.0392</td><td>0.0385</td><td>0.0401</td></tr><tr><td>Prop</td><td>0.0658</td><td>0.0647</td><td></td></tr></table></td></tr>
</table>

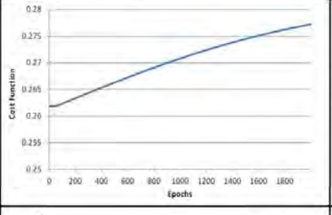
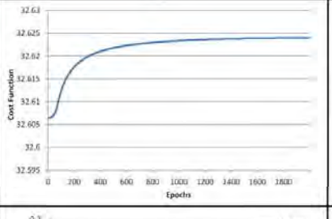
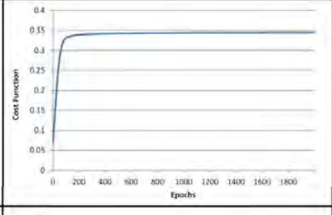
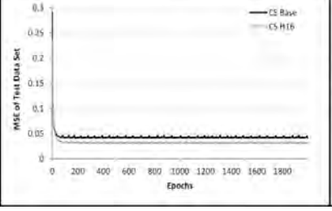
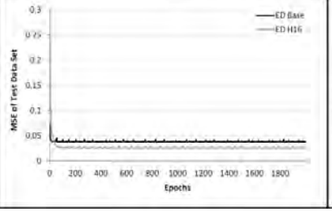
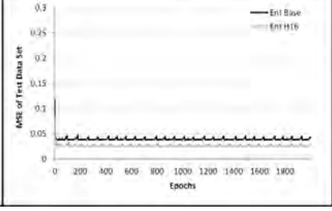
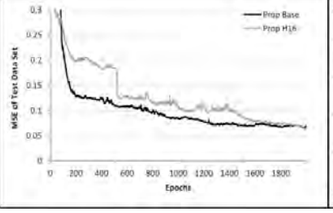
Annex 2 - Experiment T6

<table>
<tr><td>Scenario</td><td colspan="2">Unsupervised Learning
 

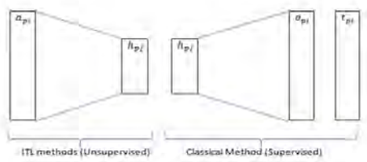
Iterations 1st # Iterations 2nd</td><td>Supervised Learning
 

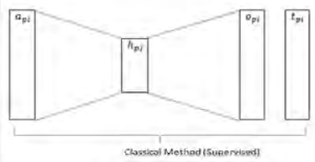
Iterations</td><td>Architecture: 24 – 16 – 24

Data: Source 1 (Normal)
Normalization: By Entrance
InitializeWeights: random
- Best combination of parameters
- Best Seed</td></tr>
<tr><td></td><td>MI Cauchy Swartz</td><td>MI Euclidean Distance</td><td>Entropy</td><td>BackPropagation</td><td>Comparative</td></tr>
<tr><td>Normal H16 Scenario</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>Description</td><td># Iterations: 2000 (1st) + 1512 (2nd)</td><td># Iterations: 1988 (1st) + 186 (2nd)</td><td># Iterations: 2000 (1st) + 43 (2nd)</td><td># Iterations: 2000</td><td></td></tr>
<tr><td></td><td colspan="3"> 
</td><td colspan="2">Error Distribution

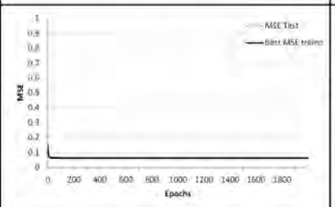
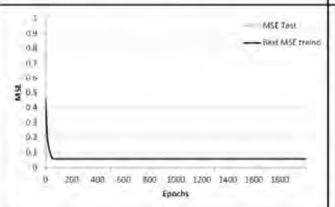
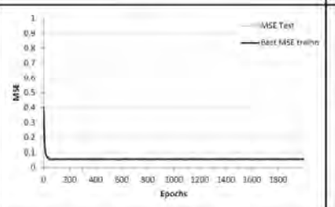
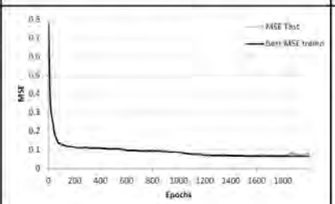
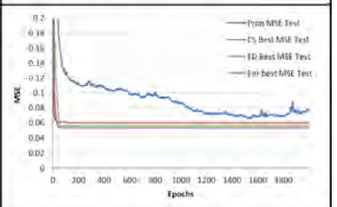
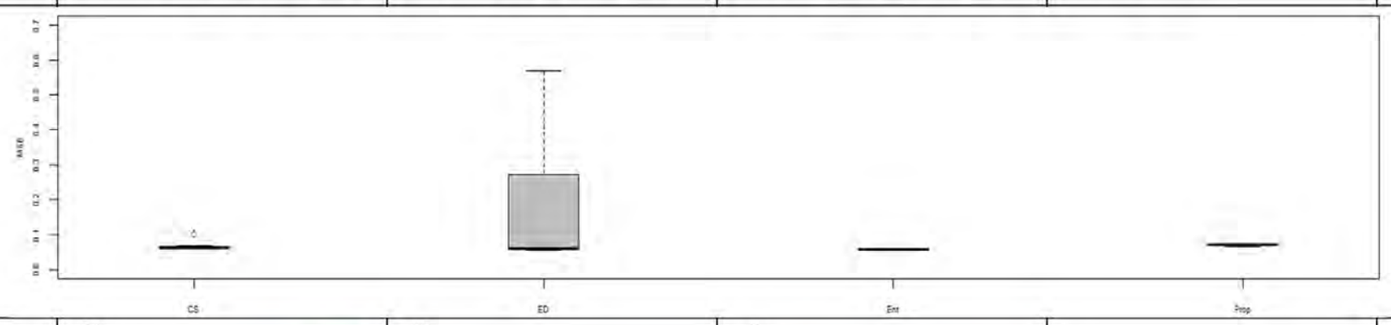
10 seeds for the Best Learning Rate Combination {c1,c2}</td></tr>
<tr><td>Cost Function (1st part)</td><td> 
</td><td> 
</td><td> 
</td><td>N.A.</td><td>N.A.</td></tr>
<tr><td>Normal H16 Vs Normal Base</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td>
<table>
<tr><th>MSE</th><th>Train</th><th>Test</th><th>Validation</th></tr>
<tr><td>CS</td><td>0.0279</td><td>0.0298</td><td>0.0308</td></tr>
<tr><td>ED</td><td>0.0248</td><td>0.0256</td><td>0.0272</td></tr>
<tr><td>Ent</td><td>0.0258</td><td>0.0255</td><td>0.0270</td></tr>
<tr><td>Prop</td><td>0.0604</td><td>0.0621</td><td></td></tr>
</table>
</td></tr>
</table>

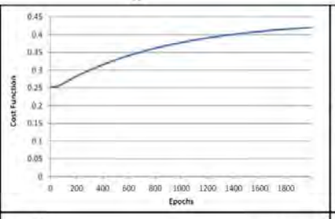
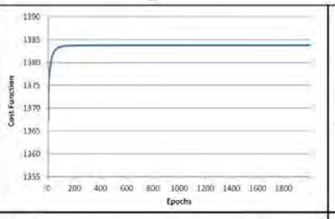
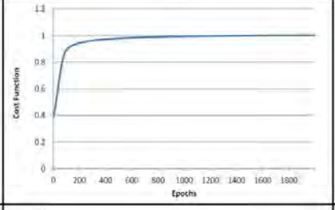
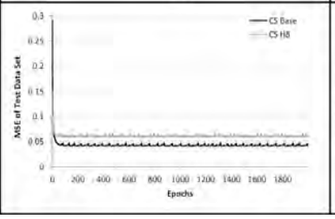
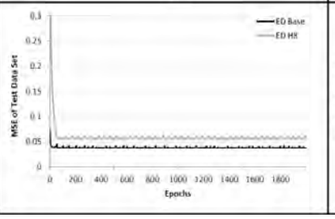
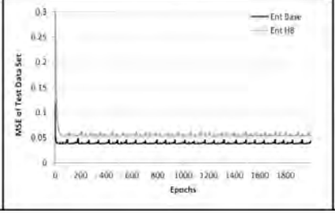
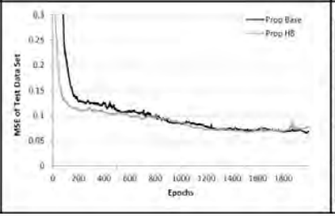
Annex 3 - Experiment T7

<table>
<tr><td>Scenario</td><td colspan="3">Unsupervised Learning
 

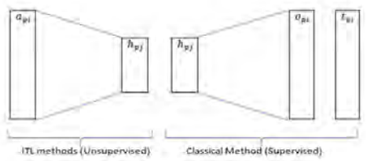
Iterations 1st # Iterations 2nd</td><td>Supervised Learning
 

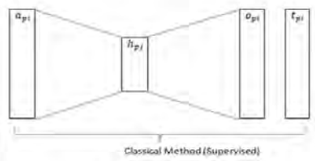
Iterations</td><td>Architecture: 24 – 8 – 24

Data: Source 1 (Normal)
Normalization: By Entrance
InitializeWeights: random
- Best combination of parameters
- Best Seed</td></tr>
<tr><td></td><td>MI Cauchy Swartz</td><td>MI Euclidean Distance</td><td>Entropy</td><td>BackPropagation</td><td>Comparative</td></tr>
<tr><td>NormalH8 Scenario</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>Description</td><td># Iterations: 2000 (1st) + 1894 (2nd)</td><td># Iterations: 816 (1st) + 102 (2nd)</td><td># Iterations: 2000 (1st) + 231 (2nd)</td><td># Iterations: 2000</td><td></td></tr>
<tr><td colspan="4"> 
</td><td>Error Distribution

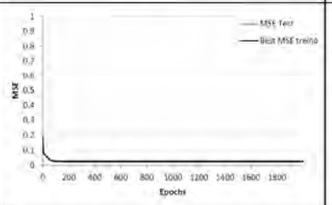
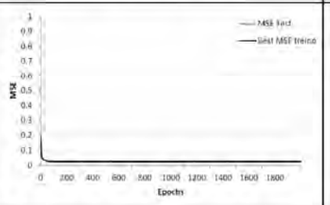
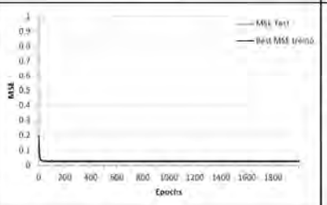
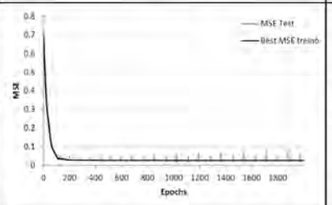
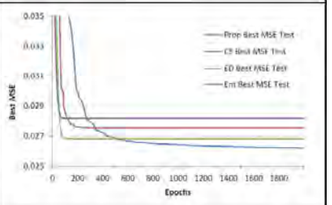
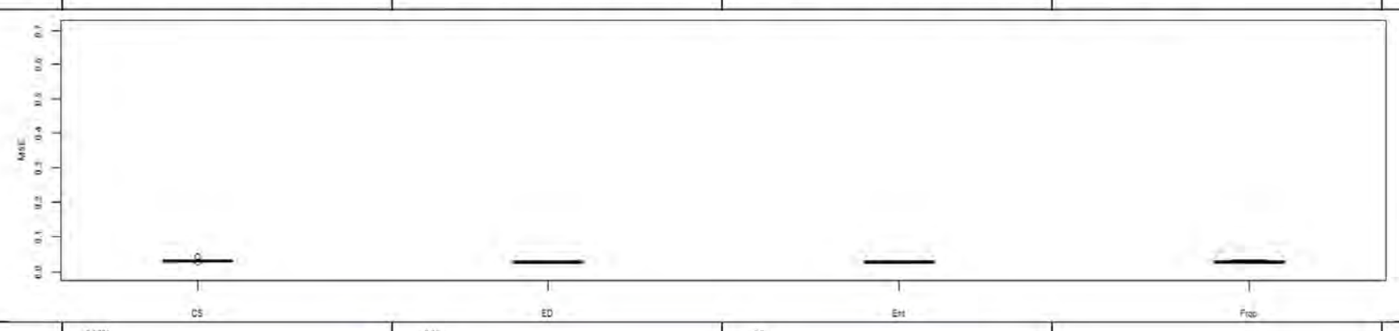
10 seeds for the Best Learning Rate Combination {c1,c2}</td></tr>
<tr><td>Cost Function (1st part)</td><td> 
</td><td> 
</td><td> 
</td><td>N.A.</td><td>N.A.</td></tr>
<tr><td>Normal H8 Vs Normal Base</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td><table><tr><th>MSE</th><th>Train</th><th>Test</th><th>Validation</th></tr><tr><td>CS</td><td>0.0610</td><td>0.0601</td><td>0.0637</td></tr><tr><td>ED</td><td>0.0561</td><td>0.0560</td><td>0.0588</td></tr><tr><td>Ent</td><td>0.0554</td><td>0.0537</td><td>0.0580</td></tr><tr><td>Prop</td><td>0.0644</td><td>0.0655</td><td></td></tr></table></td></tr>
</table>

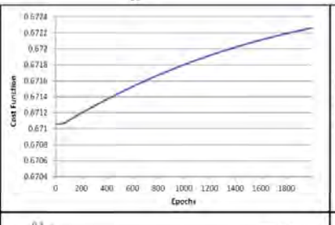
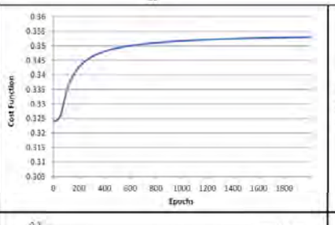
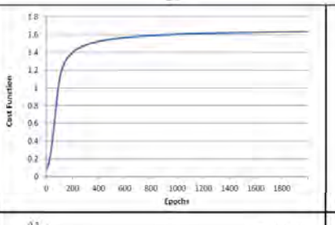
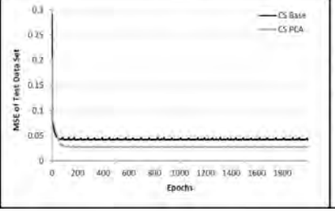
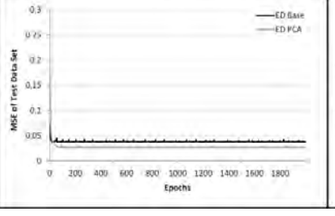
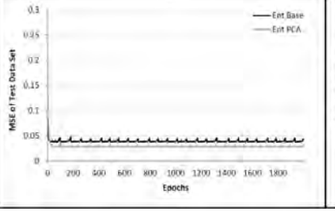
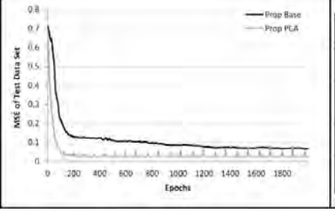
Annex 4 - Experiment T8

<table>
<tr><td>Scenario</td><td colspan="3">Unsupervised Learning
 

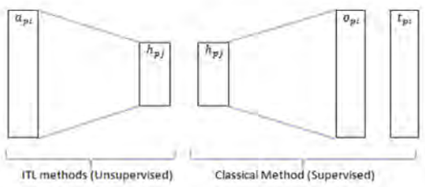
Iterations 1st # Iterations 2nd</td><td>Supervised Learning
 

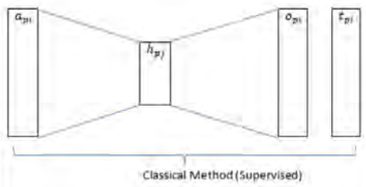
Iterations</td><td>Architecture: 24 – 12 – 24

Data: Source 1 (Normal)
Normalization: By Entrance
InitializeWeights: PCA
- Best combination of parameters
- Best Seed</td></tr>
<tr><td></td><td>MI Cauchy Swartz</td><td>MI Euclidean Distance</td><td>Entropy</td><td>BackPropagation</td><td>Comparative</td></tr>
<tr><td>Normal PCA Scenario</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>Description</td><td># Iterations: 2000 (1st) + 366 (2nd)</td><td># Iterations: 2000 (1st) + 282 (2nd)</td><td># Iterations: 2000 (1st) + 192 (2nd)</td><td># Iterations: 2000</td><td></td></tr>
<tr><td colspan="3"> 
</td><td colspan="2">Error Distribution

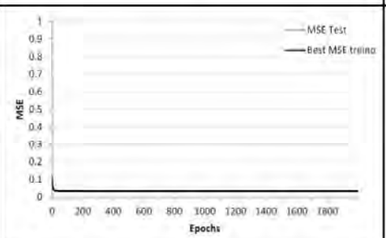
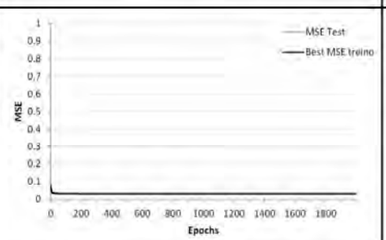
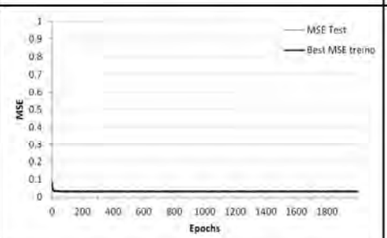
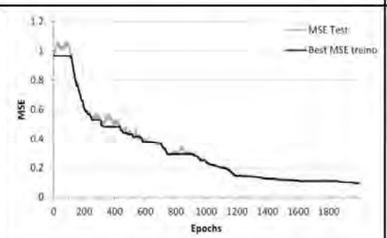
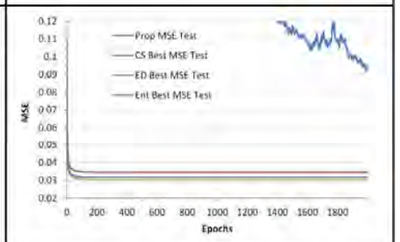
10 seeds for the Best Learning Rate Combination {c1,c2}</td></tr>
<tr><td>Cost Function (1st part)</td><td> 
</td><td> 
</td><td> 
</td><td>N.A.</td><td>N.A.</td></tr>
<tr><td>Normal PCA Vs Normal Base</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td>
<table>
<tr><th>MSE</th><th>Train</th><th>Test</th><th>Validation</th></tr>
<tr><td>CS</td><td>0.0262</td><td>0.0275</td><td>0.0293</td></tr>
<tr><td>ED</td><td>0.0255</td><td>0.0268</td><td>0.0286</td></tr>
<tr><td>Ent</td><td>0.0272</td><td>0.0282</td><td>0.0302</td></tr>
<tr><td>Prop</td><td>0.0251</td><td>0.0262</td><td>N.A.</td></tr>
</table>
</td></tr>
</table>

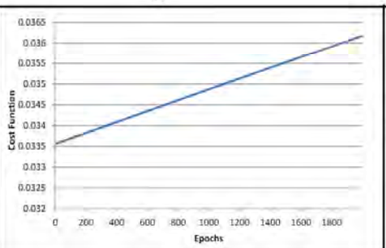
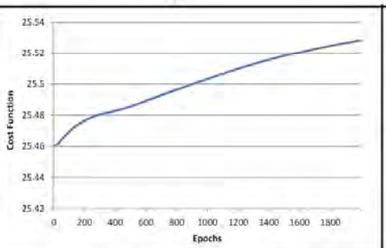
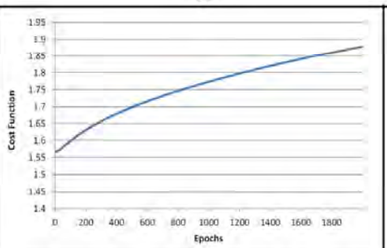
Annex 5 - Experiment T9

<table>
<tr><td>Scenario</td><td colspan="3">Unsupervised Learning
 

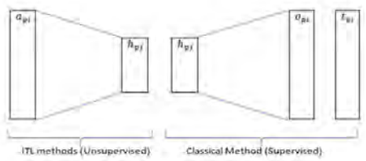
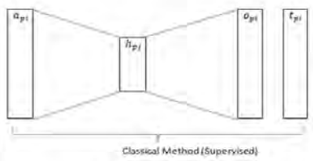
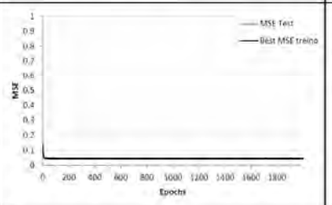
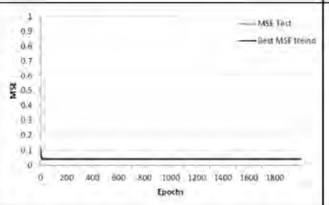
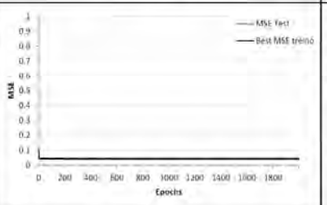
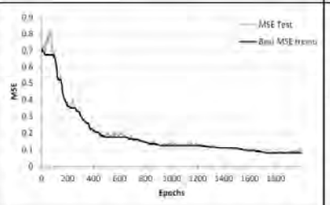
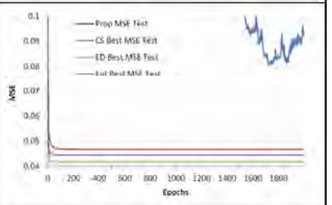
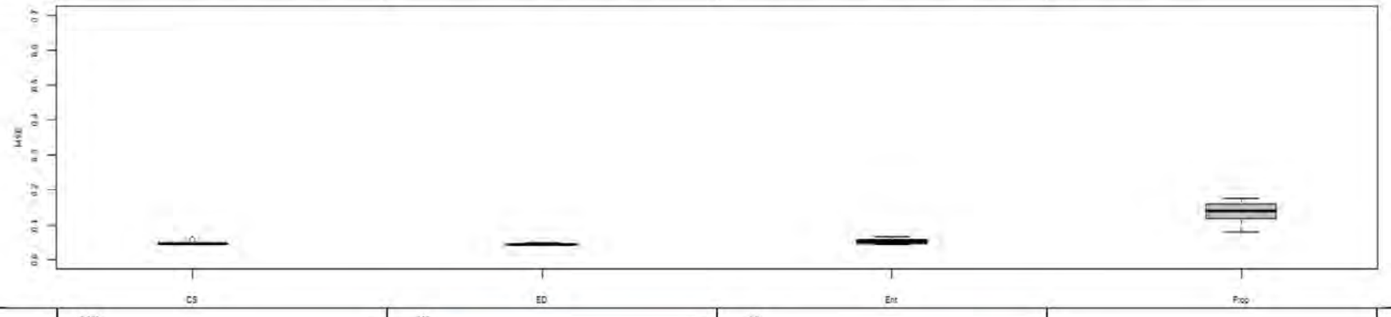
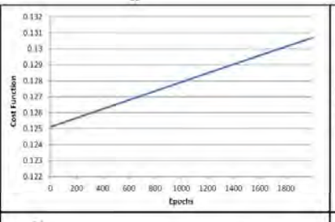
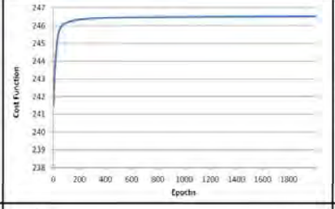
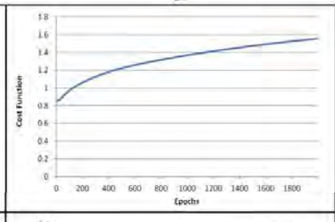
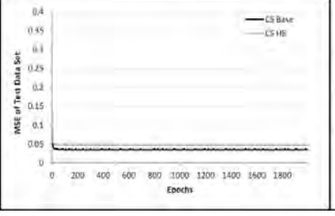
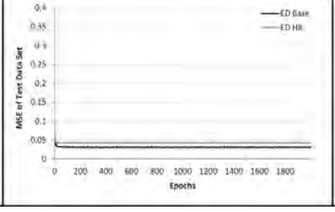
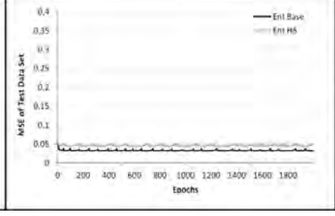
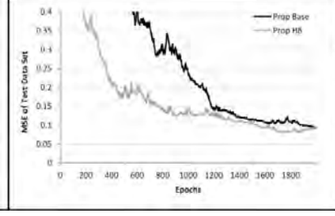
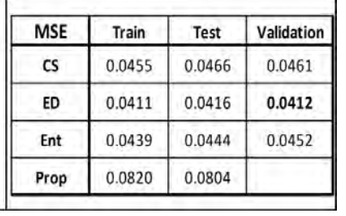
Iterations 1st # Iterations 2nd</td><td>Supervised Learning
 

Iterations</td><td>Architecture: 24 – 12 – 24

Data: Source 1+3 (Multi)
Normalization: By Entrance
InitializeWeights: random
- Best combination of parameters
- Best Seed</td></tr>
<tr><td></td><td>MI Cauchy Swartz</td><td>MI Euclidean Distance</td><td>Entropy</td><td>BackPropagation</td><td>Comparative</td></tr>
<tr><td>Multi Base Scenario</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>Description</td><td># Iterations: 2000 (1st) + 573 (2nd)</td><td># Iterations: 1999 (1st) + 1380 (2nd)</td><td># Iterations: 2000 (1st) + 467 (2nd)</td><td># Iterations: 2000</td><td></td></tr>
<tr><td colspan="4"> 
</td><td colspan="2">Error Distribution

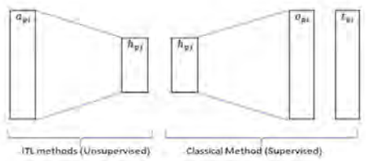
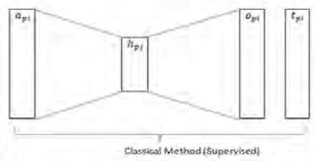
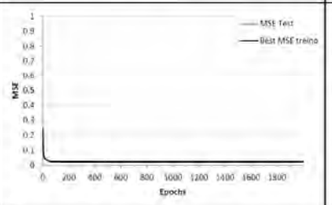
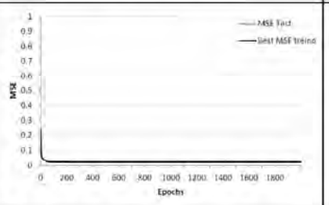
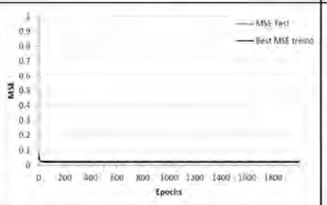
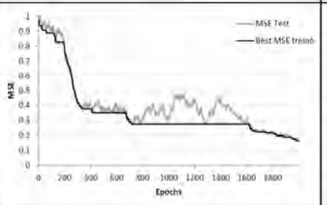
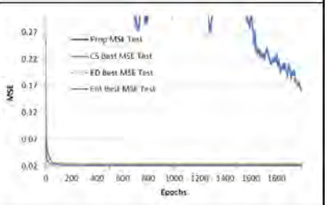
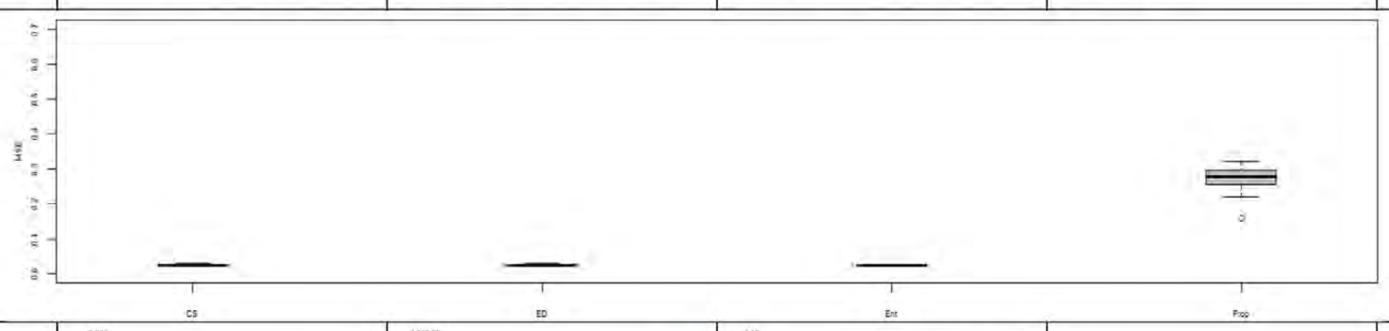
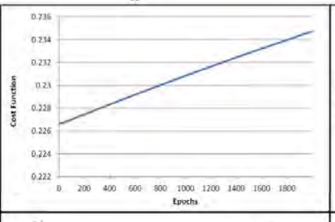
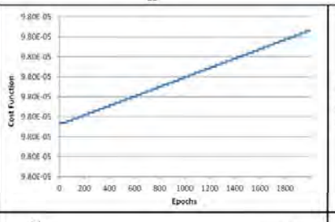
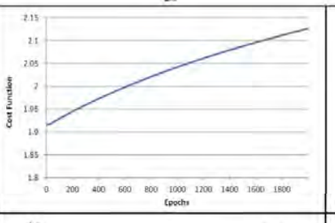
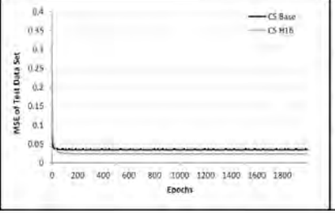
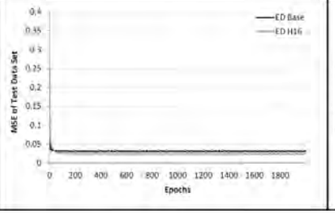
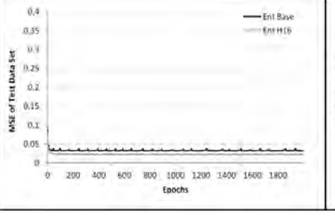
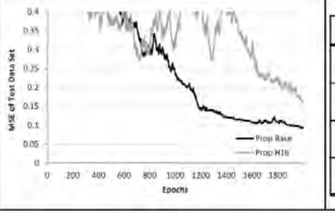
10 seeds for the Best Learning Rate Combination (c1,c2)</td></tr>
<tr><td>Cost Function (1st part)</td><td> 
</td><td> 
</td><td> 
</td><td>N.A.</td><td><table><tr><th>MSE</th><th>Train</th><th>Test</th><th>Validation</th></tr><tr><td>CS</td><td>0.0318</td><td>0.0332</td><td>0.0331</td></tr><tr><td>ED</td><td>0.0295</td><td>0.0304</td><td>0.0301</td></tr><tr><td>Ent</td><td>0.0307</td><td>0.0317</td><td>0.0319</td></tr><tr><td>Prop</td><td>0.0931</td><td>0.0919</td><td></td></tr></table></td></tr>
</table>

Annex 6 - Experiment T10

Scenario	Unsupervised Learning  # Iterations 1st # Iterations 2nd			Supervised Learning  # Iterations	**Architecture:** 24 – 8 – 24 **Data:** Source 1+3 (Multi) **Normalization:** By Entrance **InitializeWeights:** random - Best combination of parameters - Best Seed
	MI Cauchy Swartz	MI Euclidean Distance	Entropy	BackPropagation	Comparative
Multi H8 Scenario					
Description	**# Iterations:** 2000 (1st) + 525 (2nd)	**# Iterations:** 1947 (1st) + 32 (2nd)	**# Iterations:** 2000 (1st) + 556 (2nd)	**# Iterations:** 2000	
					Error Distribution 10 seeds for the Best Learning Rate Combination {c1,c2}
Cost Function (1st part)				N.A.	N.A.
Multi H8 Vs Multi Base					

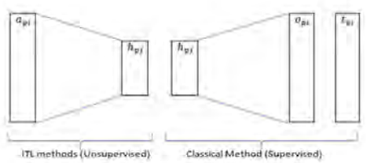
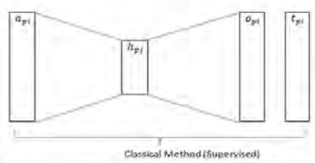
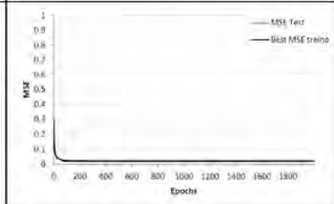
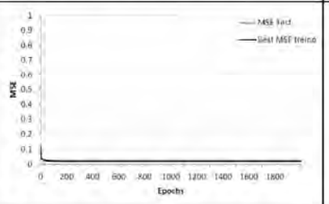
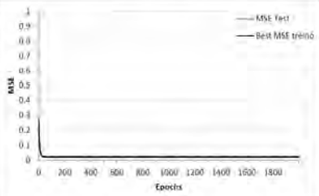
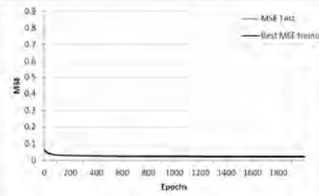
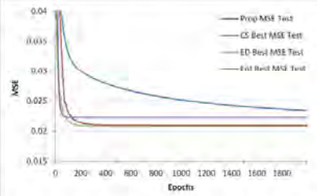
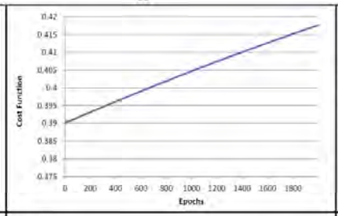
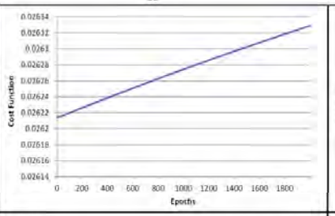
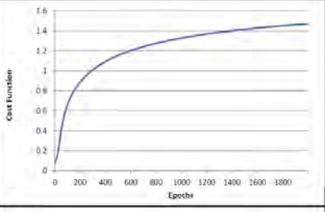
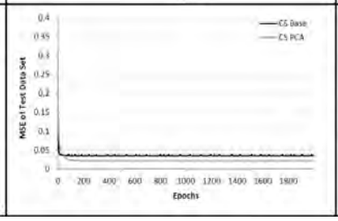
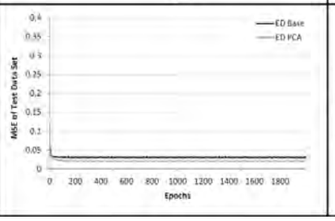
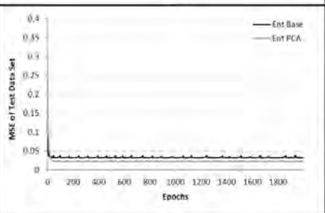
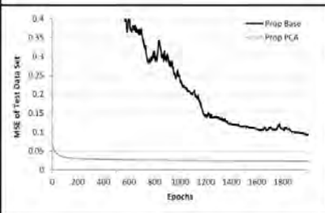
MSE	Train	Test	Validation
CS	0.0455	0.0466	0.0461
ED	0.0411	0.0416	**0.0412**
Ent	0.0439	0.0444	0.0452
Prop	0.0820	0.0804	

Annex 7 - Experiment T11

Scenario	Unsupervised Learning  # Iterations 1st # Iterations 2nd			Supervised Learning  # Iterations	**Architecture:** 24 – 16 – 24 **Data:** Source 1+3 (Multi) **Normalization:** By Entrance **InitializeWeights:** random - Best combination of parameters - Best Seed
	MI Cauchy Swartz	MI Euclidean Distance	Entropy	BackPropagation	Comparative
Multi H16 Scenario					
Description	**# Iterations:** 2000 (1st) + 1429 (2nd)	**# Iterations:** 1968 (1st) + 841 (2nd)	**# Iterations:** 2000 (1st) + 678 (2nd)	**# Iterations:** 2000	
	 				Error Distribution 10 seeds for the Best Learning Rate Combination {c1,c2}
Cost Function (1st part)				N.A.	N.A.
Multi H16 Vs Multi Base					

MSE	Train	Test	Validation
CS	0.0234	0.0240	0.0253
ED	0.0231	0.0237	0.0238
Ent	0.0214	0.0222	**0.0216**
Prop	0.1623	0.1592	

Annex 8 - Experiment T12

Scenario	Unsupervised Learning  # Iterations 1st # Iterations 2nd			Supervised Learning  # Iterations	**Architecture:** 24 – 12 – 24 **Data:** Source 1+3 (Multi) **Normalization:** By Entrance **InitializeWeights:** PCA - Best combination of parameters - Best Seed
	MI Cauchy Swartz	MI Euclidean Distance	Entropy	BackPropagation	Comparative
Multi PCA Scenario	 	 	 	 	
Description	**# Iterations:** 2000 (1st) + 1413 (2nd)	**# Iterations:** 2000 (1st) + 520 (2nd)	**# Iterations:** 2000 (1st) + 102 (2nd)	**# Iterations:** 2000	
	 				Error Distribution 10 seeds for the Best Learning Rate Combination {c1,c2}
Cost Function (1st part)	 	 	 	N.A.	N.A.
Multi PCA Vs Multi Base	 	 	 	 	

MSE	Train	Test	Validation
CS	0.0203	0.0210	0.0213
ED	0.0201	0.0208	**0.0209**
Ent	0.0214	0.0222	0.0225
Prop	0.0220	0.0234	

Analysis using descriptive statistics on the data used for the ITL networks and on the data used in the Topology problem (Power system)

PTDC/EEA-EEL/104278/2008

Report LASCA / R8

78 Abstract

The training of neural networks must consider three datasets: train, test and validation. For each experiment, the three datasets must belong to the same population. The assessment of populations' similarity for these three datasets is the main goal of this report. In this work, the Smirnov and Cramér-von Mises statistical tests were employed to three case studies. The analysis here presented is conducted on a neuron basis, meaning that the train, test and validation datasets are analyzed considering each neuron separately.

79 1 Non Parametric tests of goodness of fit

In order to infer on the similarity of the populations among the train, test and validation datasets, two non-parametric tests were applied: the **Smirnov test** and **Cramér-von Mises (CvM) test**.

These tests are non-parametric statistical procedures which use the maximum vertical distance between two cumulative functions to infer on the similarity between the two populations, see (Conover, 1980). Therefore, the tests were applied to each of the possible combinations over data sets: i) train set *vs* test set, ii) train set *vs* validation set, and iii) test set *vs* validation set.

The null hypothesis H_0 is that the two data sets arrive from similar populations. The alternative hypothesis $H_1$is that the two data sets arrive from different populations. We aim at rejecting the alternative hypothesis, which occurs for p-values higher than an alpha (significance level) considered.

This approach is the most adequate due to the following aspects:

- the data sets are of distinct dimensions (which precludes the application of non-parametric tests on several independent samples, such as the **Birnbaum-Hall test** or the **k-sample Smirnov test**);
- the assessment is made towards an unknown distribution (and not a specified one as it would be the case for using the **Kolmogorov** goodness of fit test and its variants such as **Lilliefors** or **Shapiro-Wilk**);

The assumptions considered are: the randomness of the samples, the independence of the samples, the variables are continuous and the data is ordinal.

Let us consider the first sample $(X_1, X_2, \cdots, X_n)$, which follows an unknown distribution $F(x)$, and the second sample $(Y_1, Y_2, \cdots, Y_m)$, which follows a second unknown distribution $G(x)$. The two samples are composed of different amount of elements for the general case.

Considering the following definition of empirical distribution provided by (Conover, 1980), pp. 69: "*Let $(X_1, X_2, \cdots, X_n)$ be a random sample. The empirical distribution function $S(x)$ is a function of x, which equals the fraction of X_is that are less than or equal to x for each x, $-\infty < x < +\infty$.*" Then, let be $S_1(x)$ the empirical distribution of the first sample, and $S_2(x)$ the empirical distribution of the second sample.

79.1 1.1 Formulation applied with the Smirnov test

The statistical test is formulated as specified in eq. R8/(1). The null hypothesis is that the two populations are similar, and the alternative hypothesis is that the two populations are different.

$$\begin{cases} H_0: S_1(x) = S_2(x) \\ H_1: S_1(x) \neq S_2(x) \end{cases} \qquad \textbf{eq. R8/(1)}$$

The test statistic for this test is specified as the greatest vertical distance (supreme) between the empirical distribution of the first sample and the empirical distribution of the second sample, shown in expression (2).

$$T_S = sup_x |S_1(x) - S_2(x)| \qquad \textbf{eq. R8/(2)}$$

The decision rule for a pre specified significance level α is to reject H_0 when the test statistic T_S (2) exceeds its quantile $\omega_{1-\alpha}$. For large samples, as is the case for this study, the quantiles are calculated as presented in Table R8/ 1.

Table R8/ 1 - Quantiles for a two-sided Smirnov test for different significance levels.

α	20%	10%	5%	2%	1%

$\omega_{1-\alpha}$	$1.07\sqrt{\frac{m+n}{m\cdot n}}$	$1.22\sqrt{\frac{m+n}{m\cdot n}}$	$1.36\sqrt{\frac{m+n}{m\cdot n}}$	$1.52\sqrt{\frac{m+n}{m\cdot n}}$	$1.63\sqrt{\frac{m+n}{m\cdot n}}$

79.2 1.2 Formulation applied with the Crámer-von Mises test

The statistical formulation for this test is similar to the one made for the Smirnov test. Accordingly, the formulation of the hypothesis is as specified in eq. R8/(3), where the null hypothesis tests whether the two samples arrive from similar populations, and the alternative hypothesis tests whether the two samples arrived from distinct populations.

$$\begin{cases} H_0: S_1(x) = S_2(x) \\ H_1: S_1(x) \neq S_2(x) \end{cases} \quad \textbf{eq. R8/(3)}$$

The test statistic for the **CvM** test is specified as presented in expression (4).

$$T_{CM} = \frac{m\cdot n}{(m+n)^2}\left\{\sum_{i=1}^{n}[S_1(X_i) - S_2(X_i)]^2 + \sum_{j=1}^{m}[S_1(Y_i) - S_2(Y_i)]^2\right\} \quad \textbf{eq. R8/(4)}$$

The decision rule for a pre specified significance level α is to reject H_0 when the test statistic T_{CM} (4) exceeds its quantile $\omega_{1-\alpha}$. For large samples, as is the case for this study, the quantiles are calculated as presented in Table R8/ 2.

Table R8/ 2 - Quantiles for a two-sided Crámer-von Mises test for different significance levels (for any dimension of large samples).

α	20%	10%	5%	1%	0.1%
$\omega_{1-\alpha}$	0.241	0.347	0.461	0.743	1.168

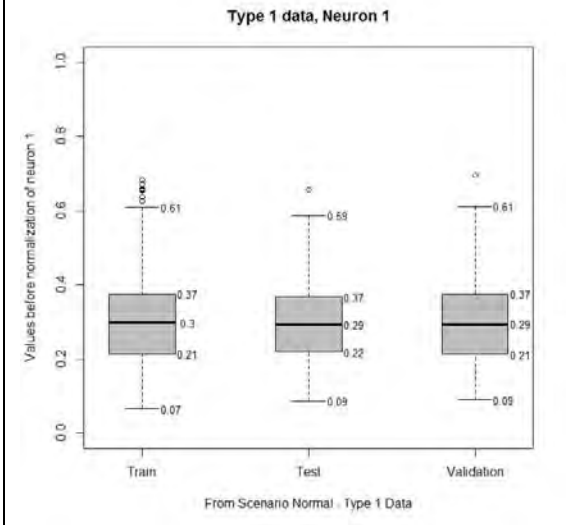
80 2 Statistical analysis of the Type 1 Data (case study “normal”)

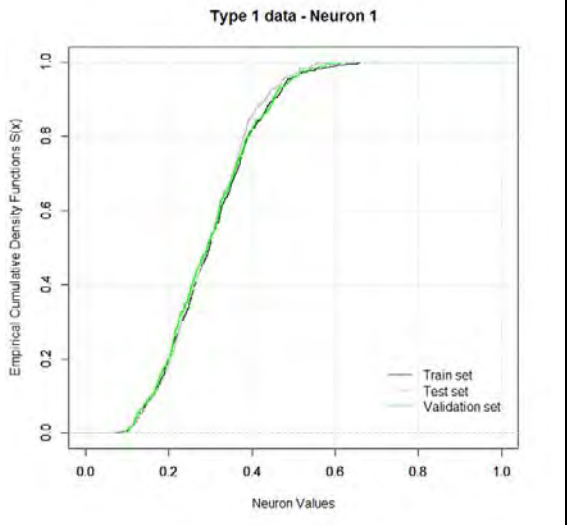
Data from wind power forecast. The train dataset contains 1000 examples, the test dataset and validation dataset contain 500 examples each. Table R8/3 presents the summary of the statistical tests conducted, where “yes” indicates the acceptance of the null hypothesis (the empirical cumulative distribution is statistically similar between the two data sets) and “no” relates to the rejection of the null hypothesis (the empirical cumulative distribution is statistically different between the two data sets), for a significance level of 5%.

Table R8/ 3 - Summary of the statistic test conducted, "yes" indicates the acceptance of the null hypothesis and "no" the rejection of the null hypothesis (α=5%).

	Train-Test:		Train-Valid:		Test-Valid	
Neuron	Smirnov	CvM	Smirnov	CvM	Smirnov	CvM
1	yes	yes	yes	yes	yes	yes
2	yes	yes	yes	yes	yes	yes
3	yes	no	yes	yes	no	yes
4	yes	yes	yes	yes	yes	yes
5	yes	yes	yes	yes	yes	yes
6	yes	yes	yes	yes	yes	yes
7	yes	yes	yes	yes	yes	yes
8	yes	yes	yes	yes	yes	yes
9	yes	yes	yes	yes	yes	yes
10	yes	yes	yes	yes	yes	yes
11	yes	yes	yes	yes	yes	yes
12	yes	no	yes	yes	yes	yes
13	no	no	yes	yes	yes	yes
14	yes	yes	yes	yes	yes	yes
15	yes	yes	yes	yes	yes	yes
16	yes	yes	yes	yes	yes	yes
17	no	no	yes	yes	yes	yes
18	yes	yes	yes	yes	yes	yes
19	no	no	yes	yes	yes	yes
20	no	no	yes	yes	yes	yes
21	yes	no	yes	yes	no	no
22	no	no	yes	yes	yes	yes
23	yes	yes	yes	yes	yes	yes
24	no	no	yes	yes	no	no

Table R8/ 4 – Results of statistical tests for case study "normal", neuron 1.

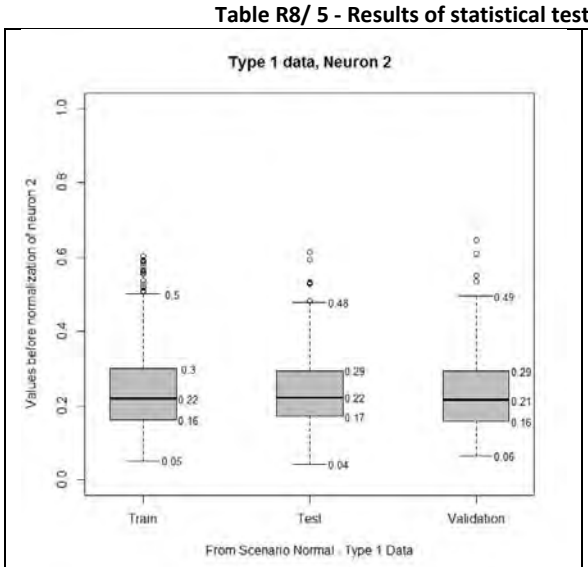


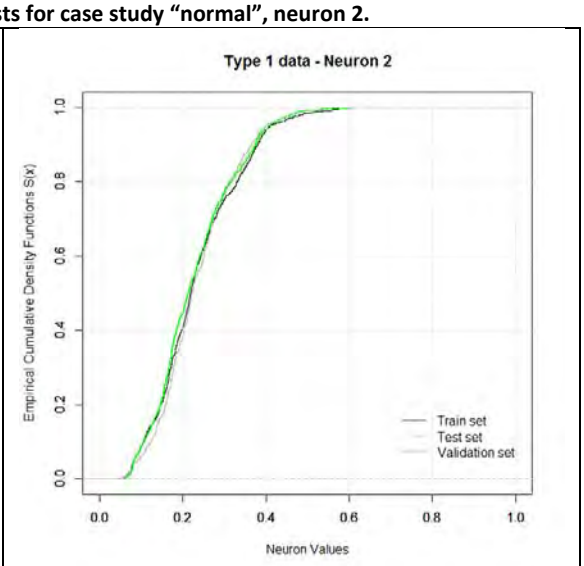


Type 1 – Neuron 1	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5990	0.8844	0.6654
CvM Test Statistic	0.1270	0.0619	0.1000

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 4).

Table R8/ 5 - Results of statistical tests for case study "normal", neuron 2.

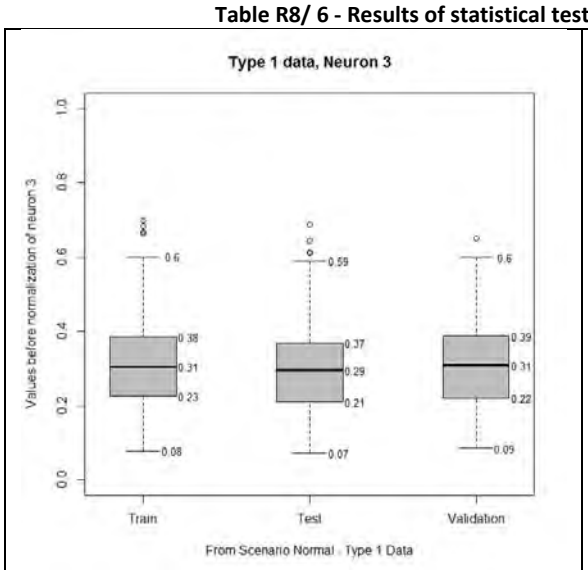


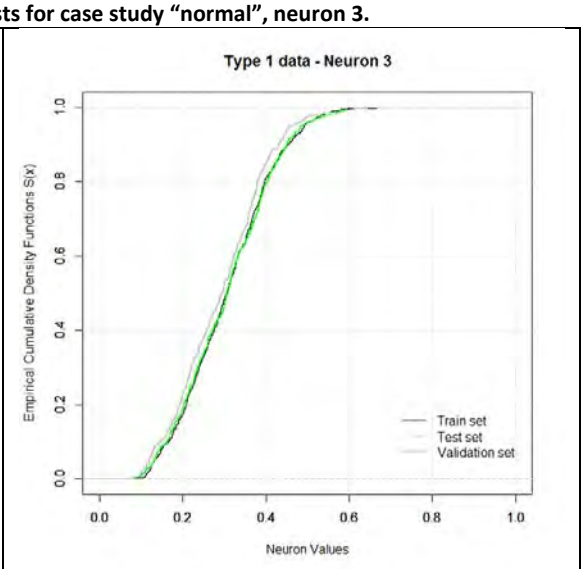


Type 1 – Neuron 2	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.3512	0.4002	0.0815
CvM Test Statistic	0.2475	0.1319	0.3902

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 5).

Table R8/ 6 - Results of statistical tests for case study "normal", neuron 3.

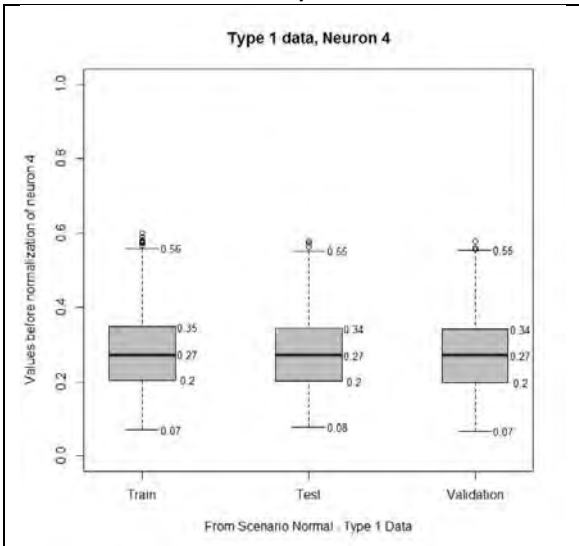


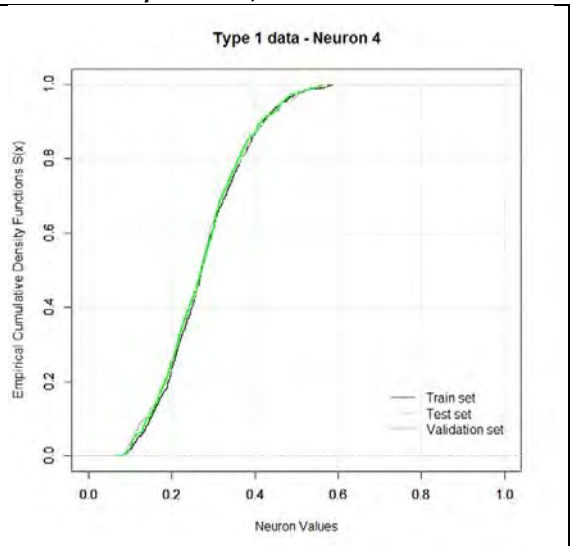


Type 1 – Neuron 3	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0837	0.9683	0.0496
CvM Test Statistic	0.5367	0.0371	0.4508

For the **Smirnov** test, the null hypothesis is accepted for the cases: Train-Test and Train-Validation, considering a significance level of 5%. The null hypothesis is rejected for the case Test-Validation, moreover the similarity of empirical cumulative density functions is not statistically significant in this case. For the **CvM** test, the null hypothesis is accepted for the cases Train-Validation and Test- Validation, considering a significance level of 5%. The null hypothesis is rejected for the case Train-Test, for a significance level of 5% (see Table R8/ 6).

Table R8/ 7 - Results of statistical tests for case study "normal", neuron 4.

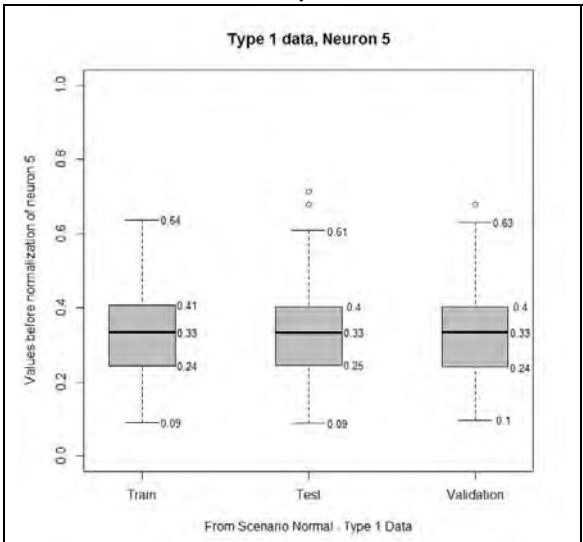


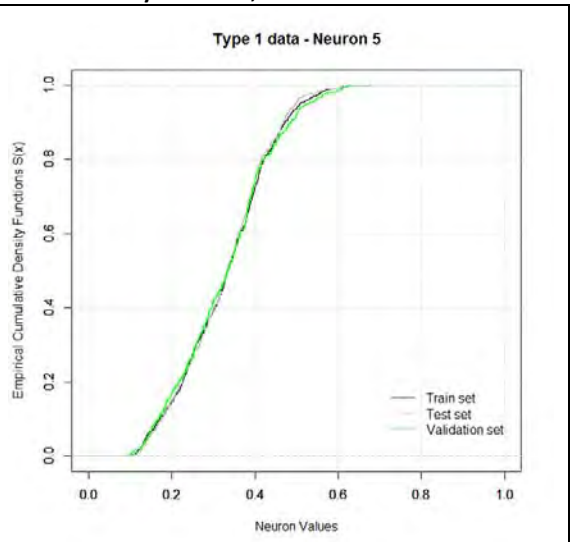


Type 1 – Neuron 4	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.6911	0.8610	0.9780
CvM Test Statistic	0.0636	0.0710	0.0367

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 7).

Table R8/ 8 - Results of statistical tests for case study "normal", neuron 5.

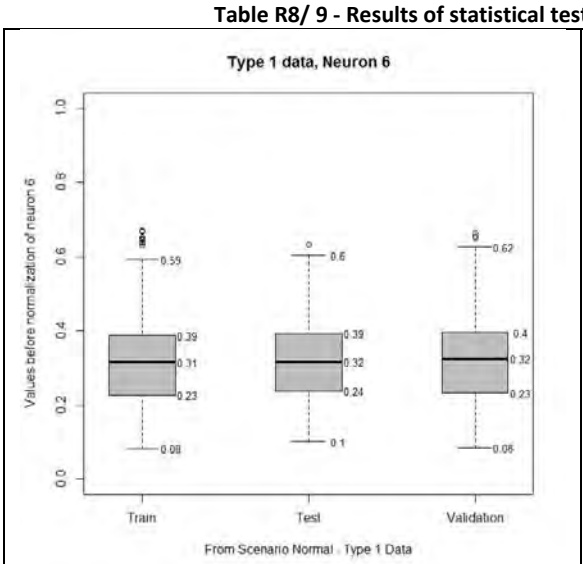


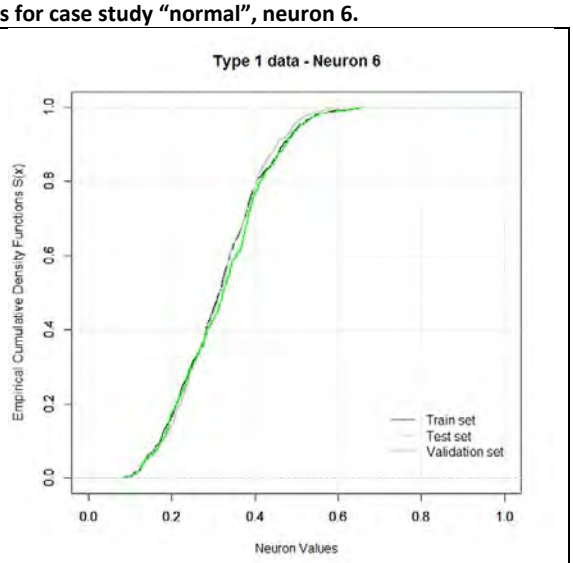


Type 1 – Neuron 5	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.9563	0.9251	0.7699
CvM Test Statistic	0.0430	0.0507	0.0737

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 8).

Table R8/ 9 - Results of statistical tests for case study "normal", neuron 6.

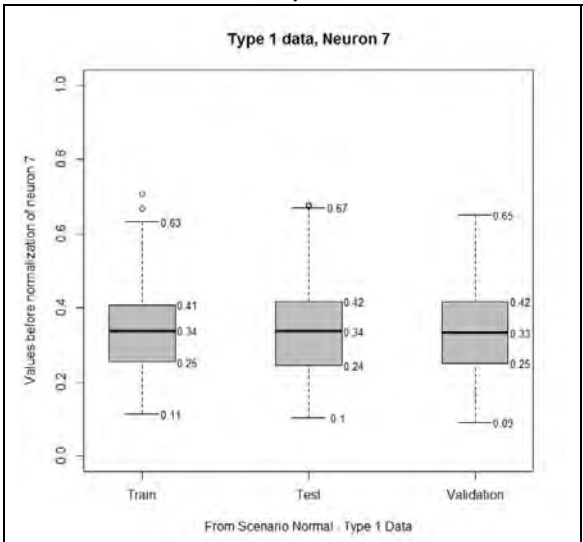


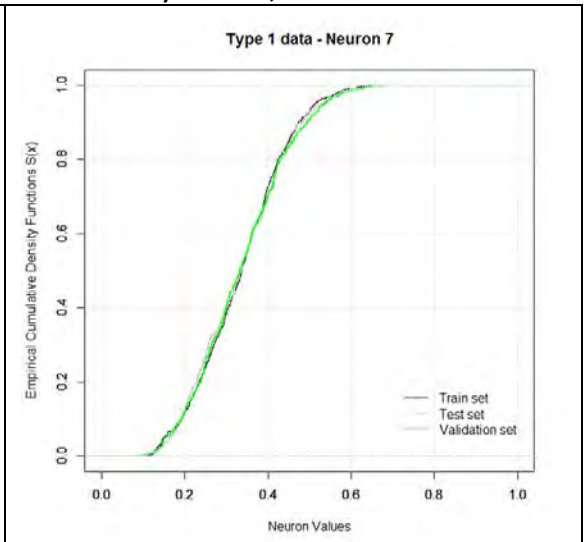


Type 1 – Neuron 6	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.7515	0.1813	0.4595
CvM Test Statistic	0.0854	0.2058	0.1524

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (Table R8/ 9).

Table R8/ 10 - Results of statistical tests for case study "normal", neuron 7.

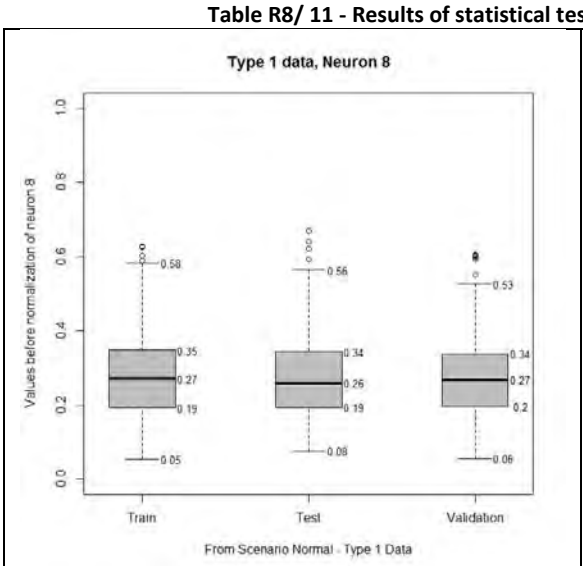


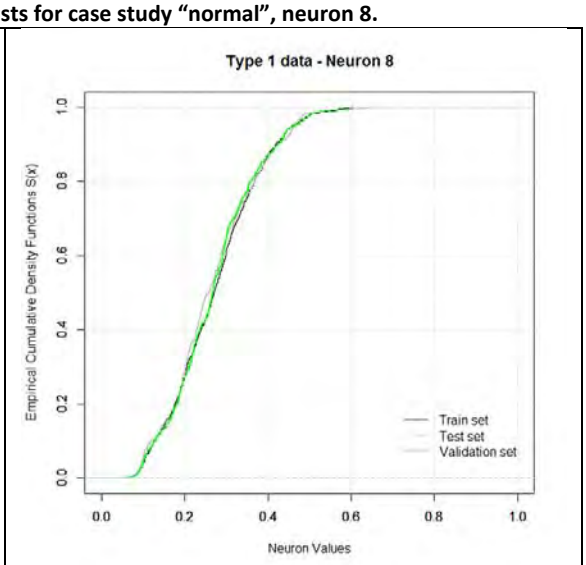


Type 1 – Neuron 7	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.4531	0.7807	0.9022
CvM Test Statistic	0.0984	0.0759	0.0484

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (Table R8/ 10).

Table R8/ 11 - Results of statistical tests for case study "normal", neuron 8.

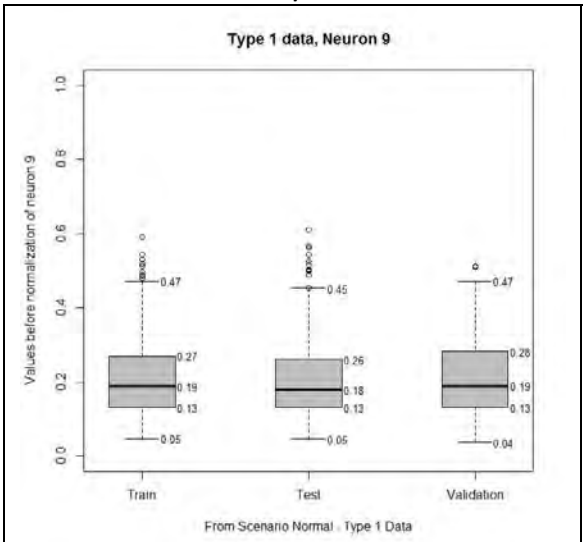


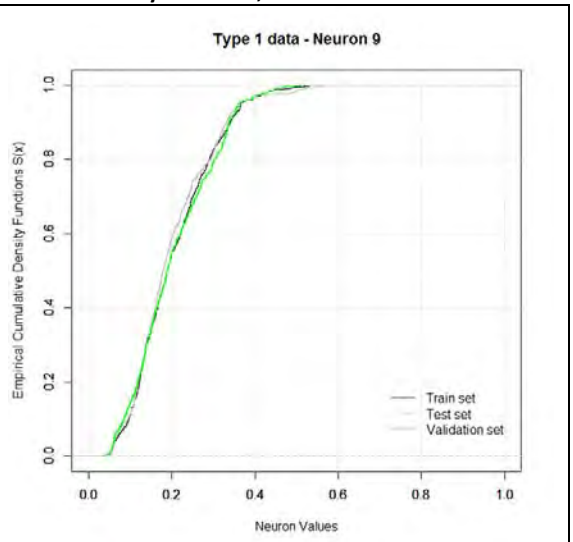


Type 1 – Neuron 8	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.1303	0.4262	0.2574
CvM Test Statistic	0.2007	0.0895	0.1160

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 11).

Table R8/ 12 - Results of statistical tests for case study "normal", neuron 9.

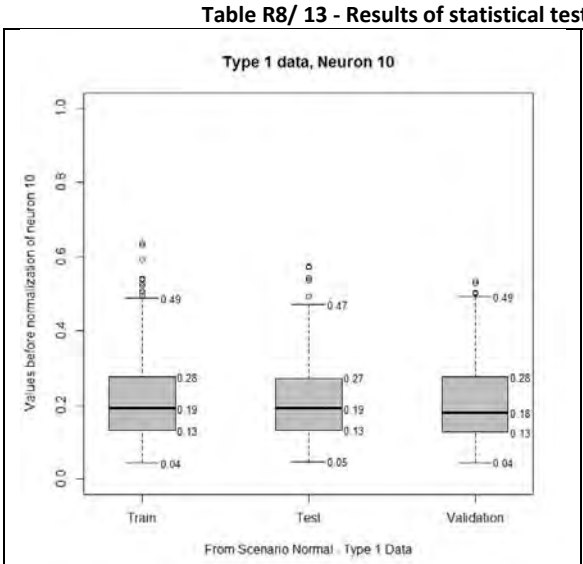


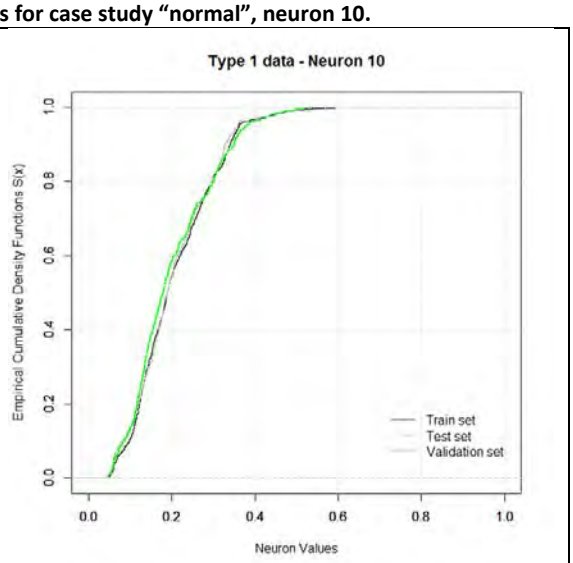


Type 1 – Neuron 9	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.2854	0.5095	0.2574
CvM Test Statistic	0.2067	0.0880	0.2369

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 12).

Table R8/ 13 - Results of statistical tests for case study "normal", neuron 10.

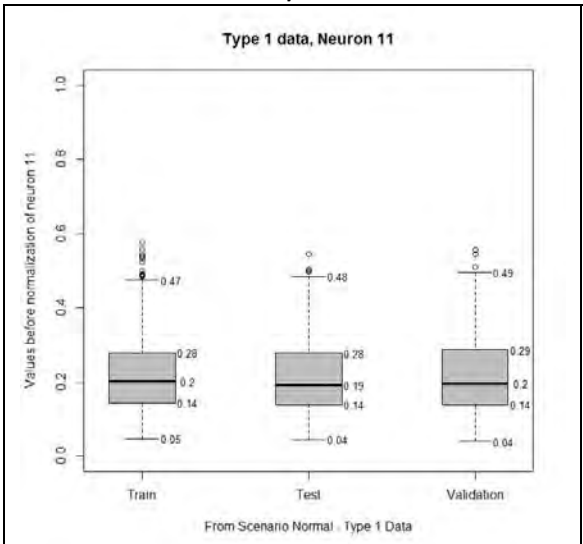


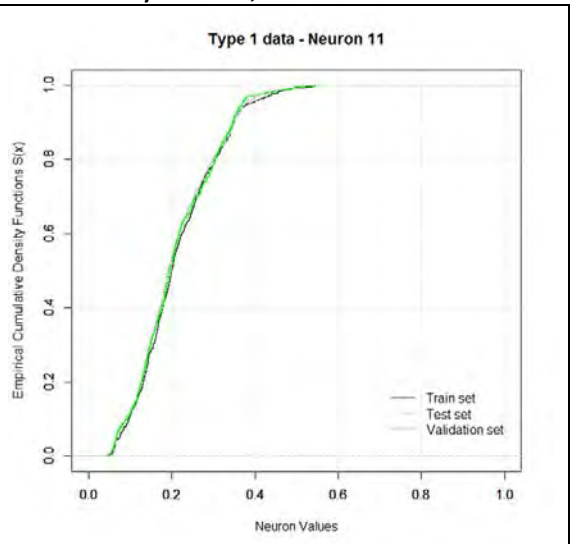


Type 1 – Neuron 10	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.7515	0.1541	0.1294
CvM Test Statistic	0.0810	0.3312	0.2933

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 13).

Table R8/ 14 - Results of statistical tests for case study "normal", neuron 11.

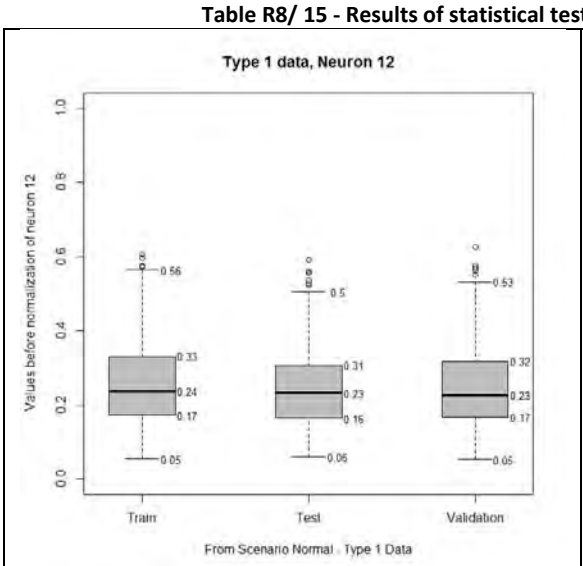


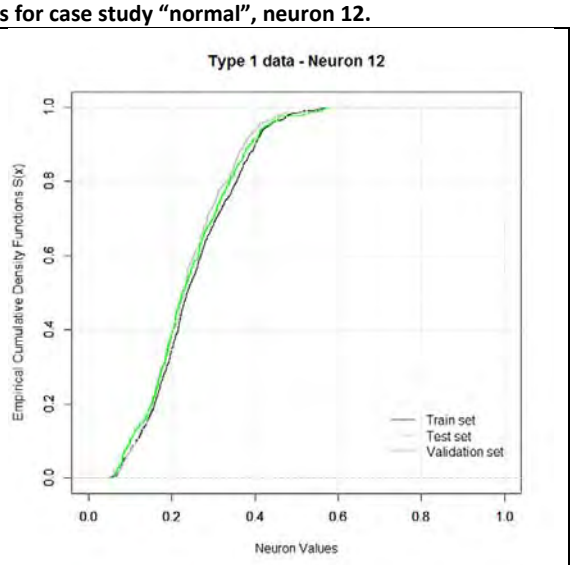


Type 1 – Neuron 11	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.3512	0.9059	0.9600
CvM Test Statistic	0.1412	0.0753	0.0389

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 14).

Table R8/ 15 - Results of statistical tests for case study "normal", neuron 12.

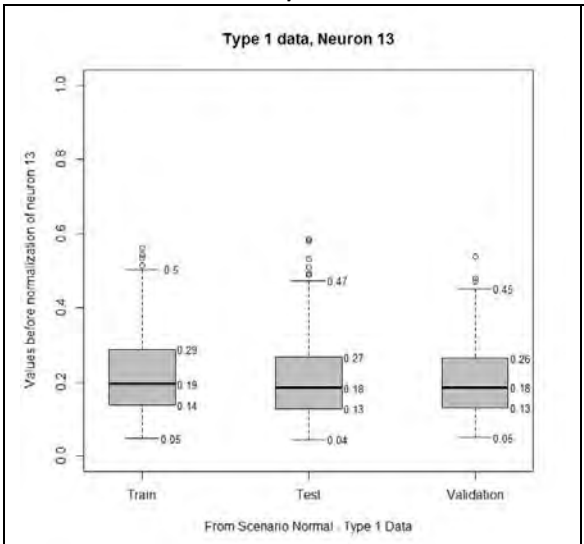


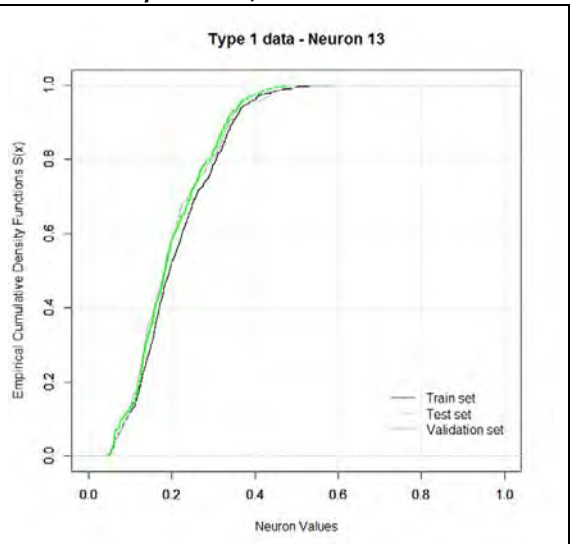


Type 1 – Neuron 12	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.1673	0.1962	0.8632
CvM Test Statistic	0.4616	0.2771	0.1039

For the **Smirnov** test, the null hypothesis is accepted in all cases, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered. For the **CvM** test, the null hypothesis is accepted for the cases Train-Validation and Test-Validation considering a significance level of 5%. The null hypothesis is rejected for the case Train-Test (see Table R8/ 15).

Table R8/ 16 - Results of statistical tests for case study "normal", neuron 13.

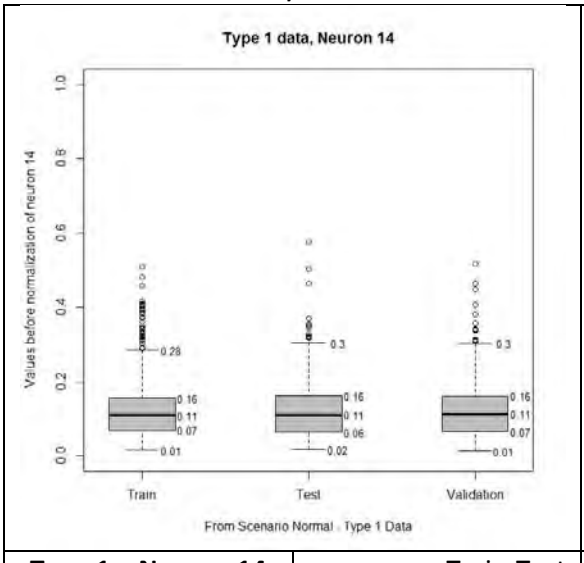


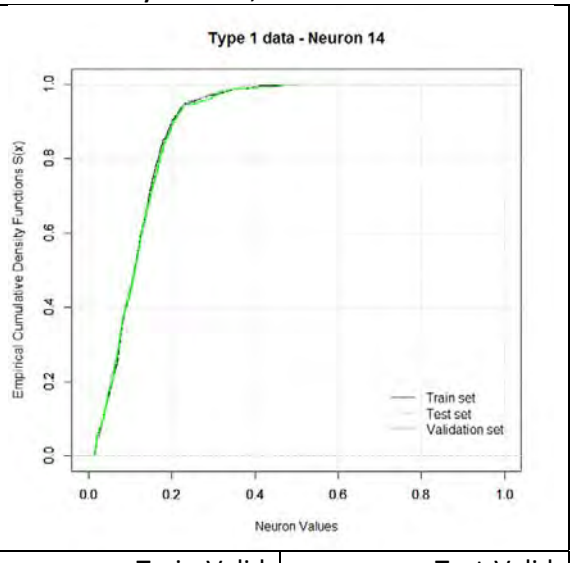


Type 1 – Neuron 13	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0203	0.1096	0.9347
CvM Test Statistic	0.6430	0.4461	0.0702

Both tests return the acceptance of the cases Train-Validation and Test-Validation, considering a significance level of 5%. Both tests rejected the null hypothesis for the case Train-Test, from where the similarity of the two populations cannot be inferred (see Table R8/ 16).

Table R8/ 17 - Results of statistical tests for case study "normal", neuron 14.

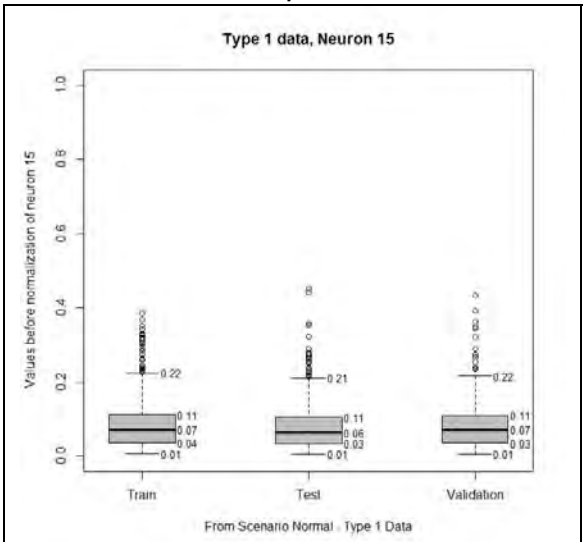


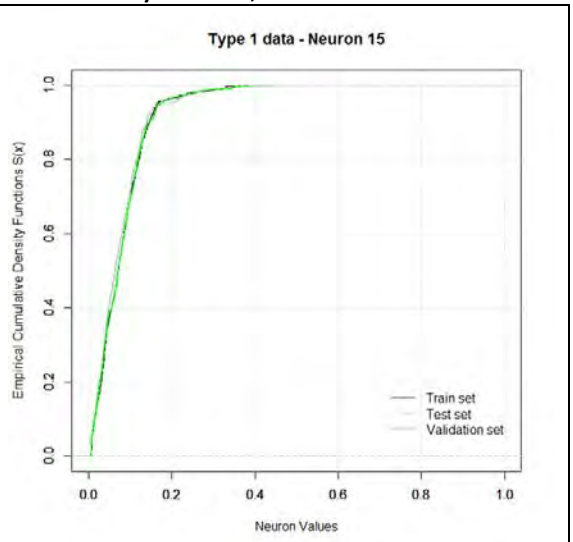


Type 1 – Neuron 14	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.7807	0.8088	0.9895
CvM Test Statistic	0.0863	0.0590	0.0329

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 17).

Table R8/ 18 - Results of statistical tests for case study "normal", neuron 15.

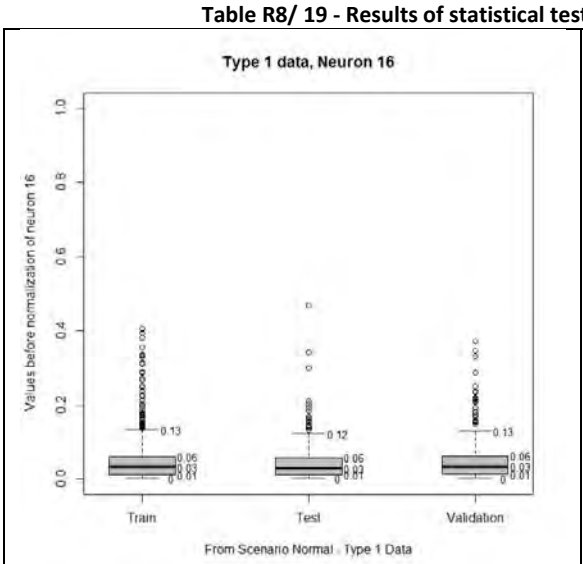


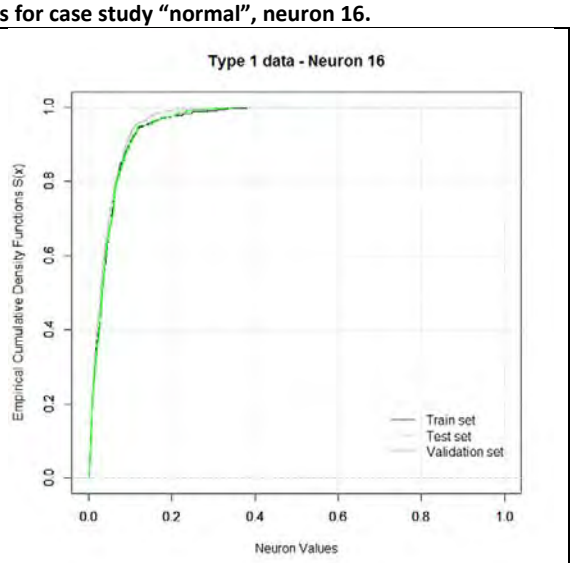


Type 1 – Neuron 15	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.1541	0.9419	0.1294
CvM Test Statistic	0.2297	0.0330	0.1927

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 18).

Table R8/ 19 - Results of statistical tests for case study "normal", neuron 16.

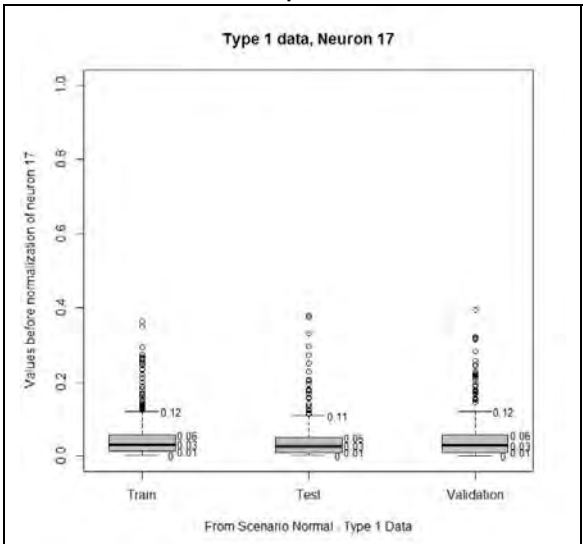


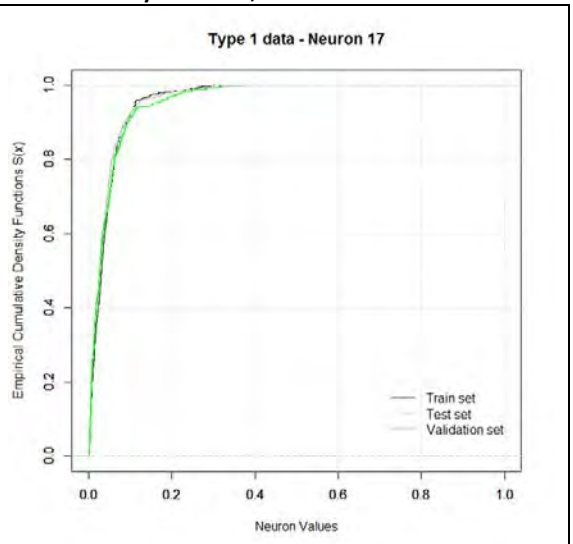


Type 1 – Neuron 16	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.3063	0.7807	0.2262
CvM Test Statistic	0.2447	0.0775	0.3193

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 19).

Table R8/ 20 - Results of statistical tests for case study “normal”, neuron 17.

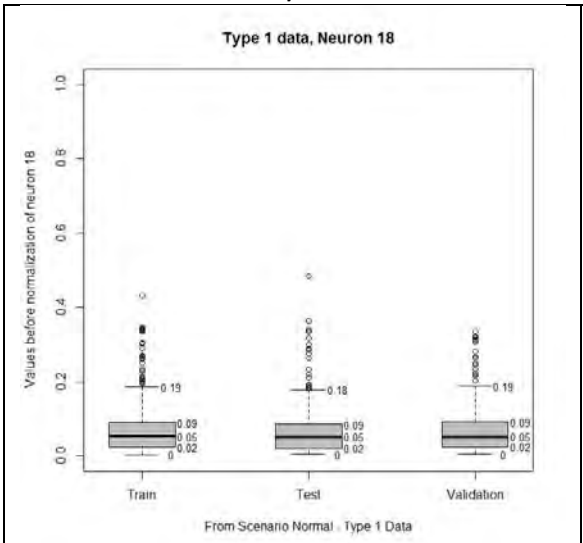


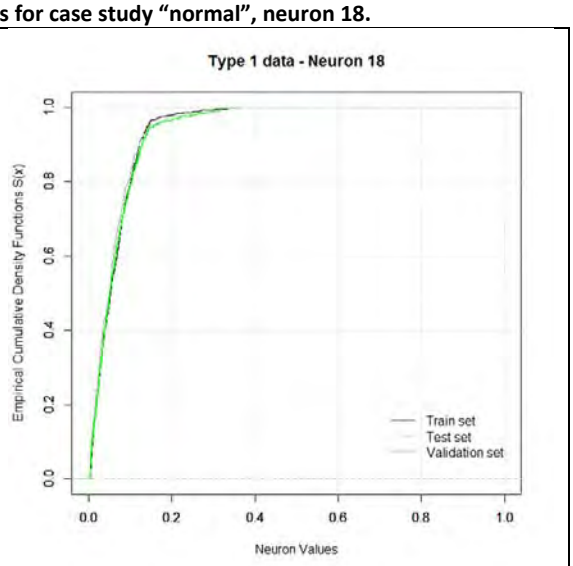


Type 1 – Neuron 17	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0425	0.1962	0.2917
CvM Test Statistic	0.5387	0.2775	0.1428

Both tests return the acceptance of the null hypothesis for the cases Train-Validation and Test-Validation, considering a significance level of 5%. Both tests replied with the rejection of the null hypothesis for the case Train-Test, from where the similarity of these two populations cannot be ascertain (see Table R8/ 20).

Table R8/ 21 - Results of statistical tests for case study “normal”, neuron 18.

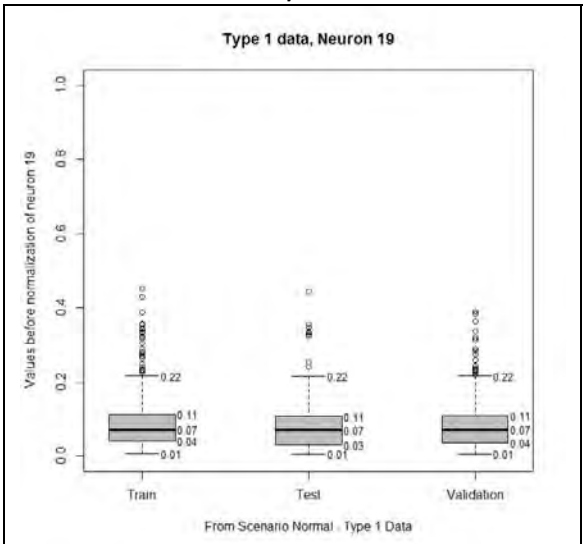


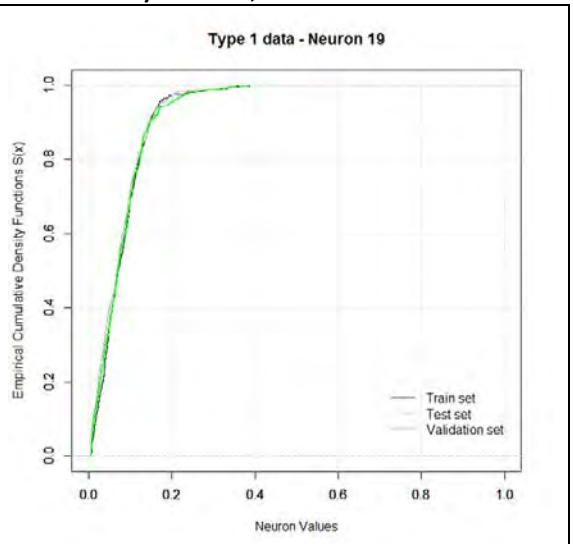


Type 1 – Neuron 18	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.1096	0.8357	0.5085
CvM Test Statistic	0.3162	0.0472	0.2197

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 21).

Table R8/ 22 - Results of statistical tests for case study "normal", neuron 19.

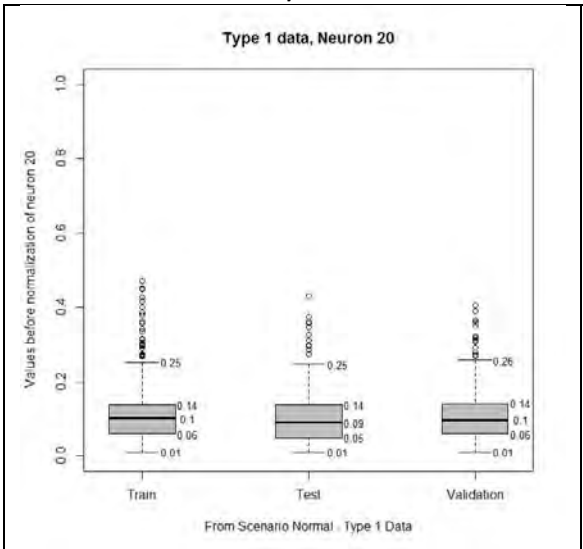


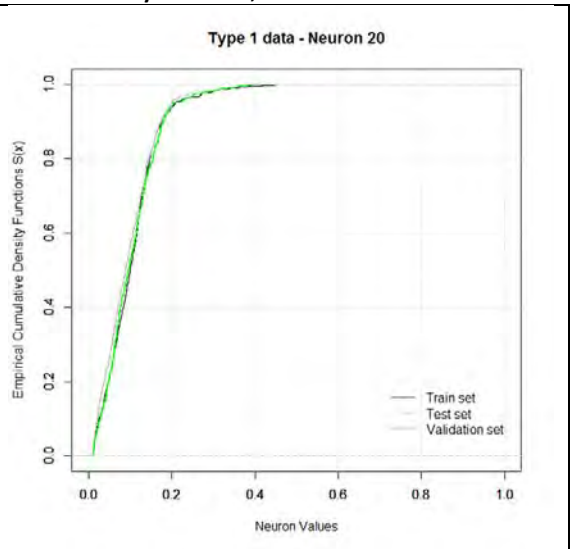


Type 1 – Neuron 19	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0181	0.3752	0.4595
CvM Test Statistic	0.5278	0.1301	0.1545

Both tests replied with the acceptance of the null hypothesis for the cases Train-Validation and Test-Validation, considering a significance level of 5%. Both tests replied with the rejection of the null hypothesis for the case Train-Test, indicating the non-similarity of these two populations (see Table R8/ 22).

Table R8/ 23 - Results of statistical tests for case study "normal", neuron 20.

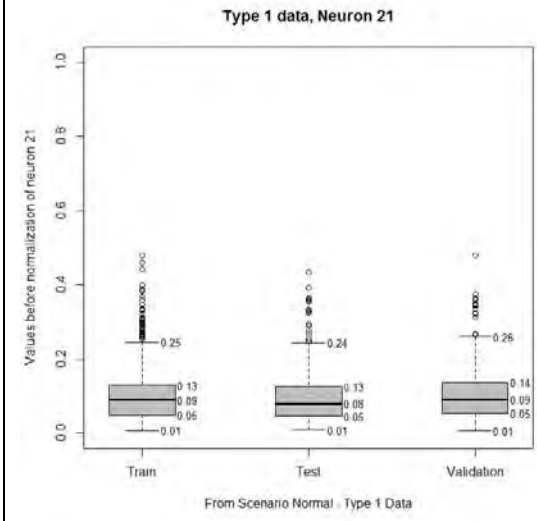


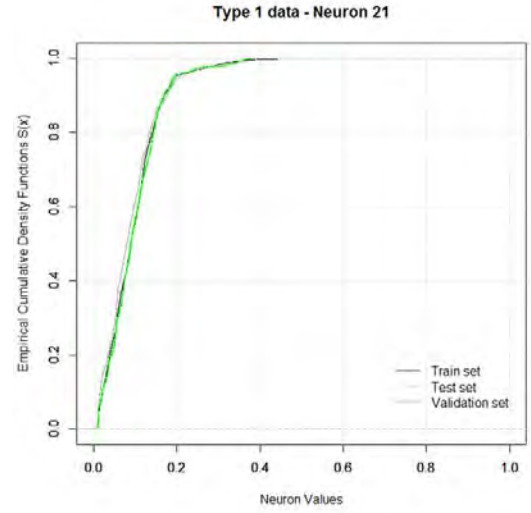


Type 1 – Neuron 20	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0470	0.4809	0.1497
CvM Test Statistic	0.5842	0.1144	0.3665

Both tests return the acceptance of the null hypothesis for the cases Train-Validation and Test-Validation, considering a significance level of 5%. Both tests returned the rejection of the null hypothesis for the case Train-Test, indicating the non-similarity of these populations (see Table R8/ 23).

Table R8/ 24 - Results of statistical tests for case study "normal", neuron 21.

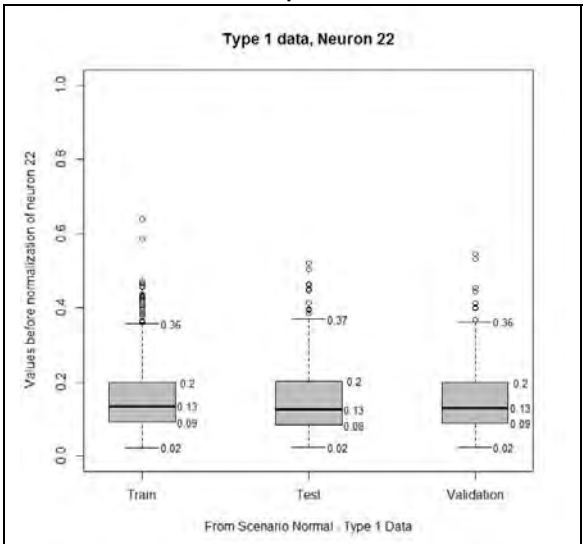


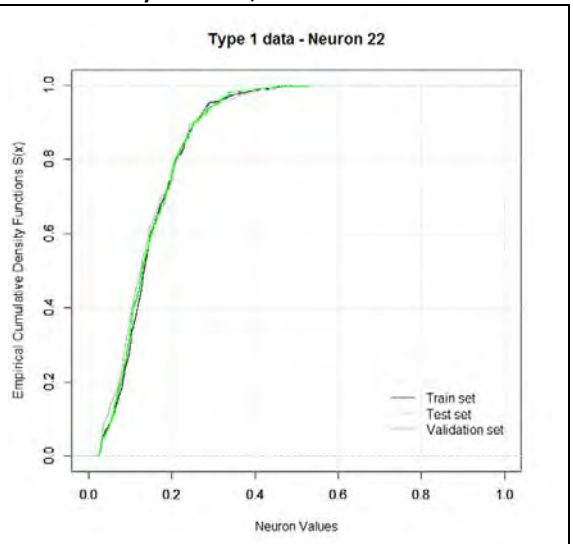


Type 1 – Neuron 21	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0631	0.5686	0.0496
CvM Test Statistic	0.4614	0.0876	0.5855

For the **Smirnov** test, the null hypothesis is accepted for the cases Train-Test and Train-Validation. This test rejected the null hypothesis for the case Test-Validation. The **CvM** test only accepted the null hypothesis for the case Train-Validation (see Table R8/ 24).

Table R8/ 25 - Results of statistical tests for case study "normal", neuron 22.

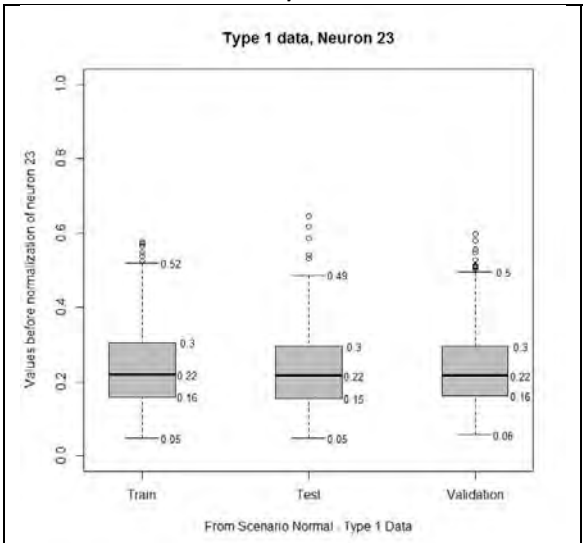


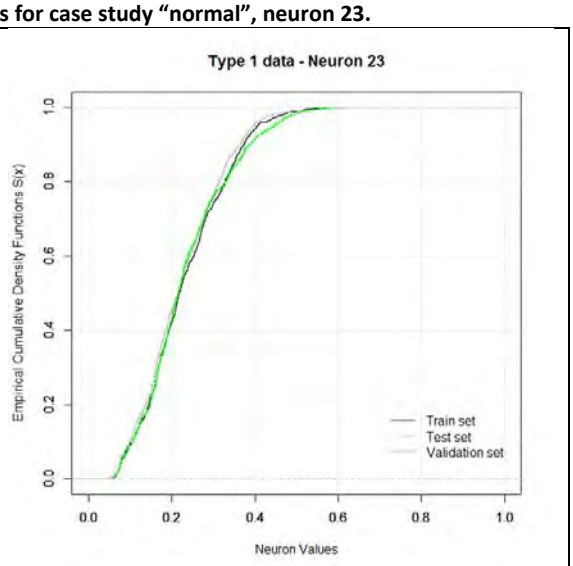


Type 1 – Neuron 22	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0425	0.4262	0.5085
CvM Test Statistic	0.5151	0.1138	0.1785

Both tests return the acceptance of the null hypothesis for the cases Train-Validation and Test-Validation, considering a significance level of 5%. Both tests returned the rejection of the null hypothesis for the case Train-Test, indicating the non-similarity of these two populations (see Table R8/ 25).

Table R8/ 26 - Results of statistical tests for case study "normal", neuron 23.

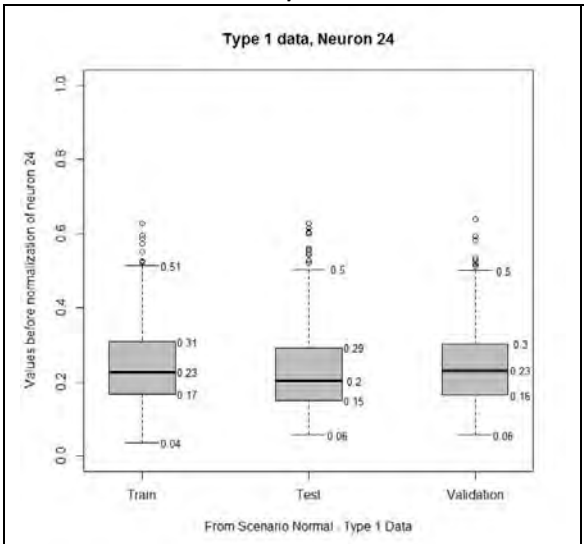


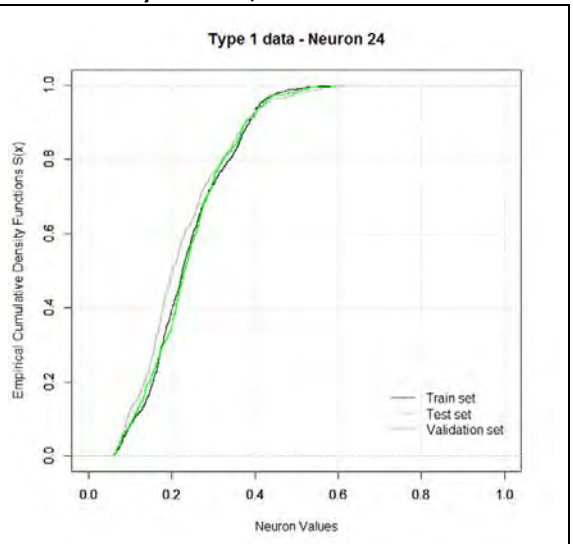


Type 1 – Neuron 23	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5387	0.8357	0.5596
CvM Test Statistic	0.1567	0.0804	0.1463

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 26).

Table R8/ 27 - Results of statistical tests for case study “normal”, neuron 24.





Type 1 – Neuron 24	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0017	0.3063	0.0000
CvM Test Statistic	1.1602	0.1371	1.1341

Both tests returned the rejection of the null hypothesis for the cases Train-Test and Test-Validation, and the acceptance of the null hypothesis for the case Train-Validation, with a significance level of 5%. For this neuron, the similarity among data sets can only be stated for the case Train-Validation (see Table R8/ 27).

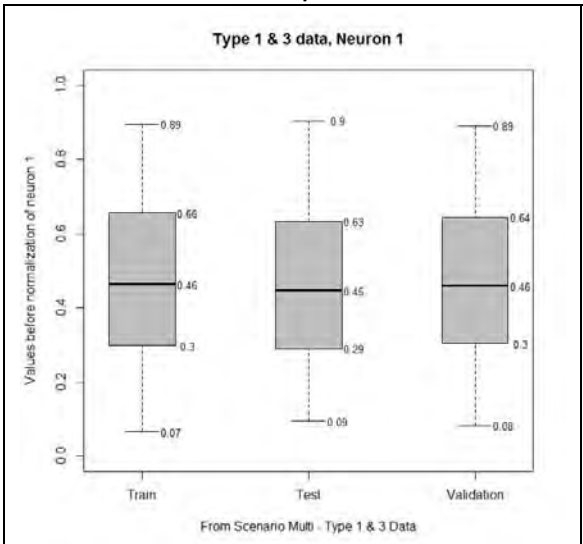
81 3 ITL networks: Data of type 1 & 3 (case study “*multi*”)

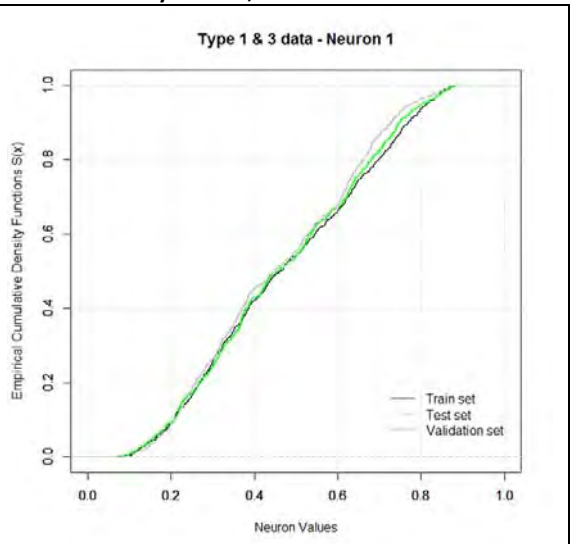
The train dataset contains 1000 examples, the test dataset and validation dataset contain 500 examples each. This section presents a summary analysis on these three data sets. Each set contains data from types 1 and 3, organized alternately (one example “type 1”, followed by one example “type 3”, followed by an example “type1”, and so on).

Table R8/ 28 - Summary of the statistic test conducted, "yes" indicates the acceptance of the null hypothesis and "no" the rejection of the null hypothesis (α=5%).

	Train-Test:		Train-Valid:		Test-Valid	
Neuron	Smirnov	CvM	Smirnov	CvM	Smirnov	CvM
1	yes	yes	yes	yes	yes	yes
2	yes	yes	yes	yes	yes	yes
3	yes	yes	yes	yes	yes	yes
4	yes	yes	yes	yes	yes	yes
5	yes	yes	yes	yes	yes	yes
6	yes	yes	yes	yes	yes	yes
7	yes	yes	yes	yes	yes	yes
8	yes	yes	yes	yes	yes	yes
9	yes	yes	yes	yes	yes	yes
10	yes	yes	yes	yes	yes	yes
11	yes	yes	yes	yes	yes	yes
12	yes	yes	yes	yes	yes	yes
13	yes	yes	yes	yes	yes	yes
14	yes	yes	yes	yes	yes	yes
15	yes	yes	yes	yes	yes	yes
16	no	yes	yes	yes	yes	yes
17	yes	yes	yes	yes	yes	yes
18	yes	yes	yes	yes	yes	yes
19	yes	yes	yes	yes	yes	yes
20	yes	yes	no	no	yes	yes
21	yes	yes	yes	yes	yes	yes
22	yes	yes	yes	yes	yes	yes
23	yes	yes	yes	yes	yes	yes
24	yes	yes	yes	yes	yes	yes

Table R8/ 29 - Results of statistical tests for case study "multi", neuron 1.

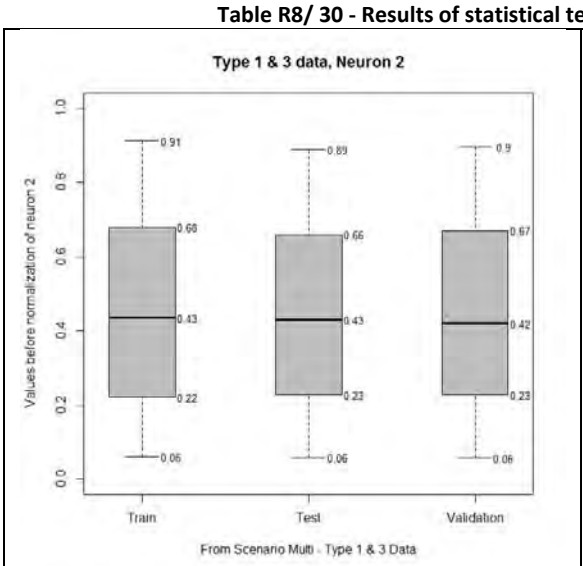


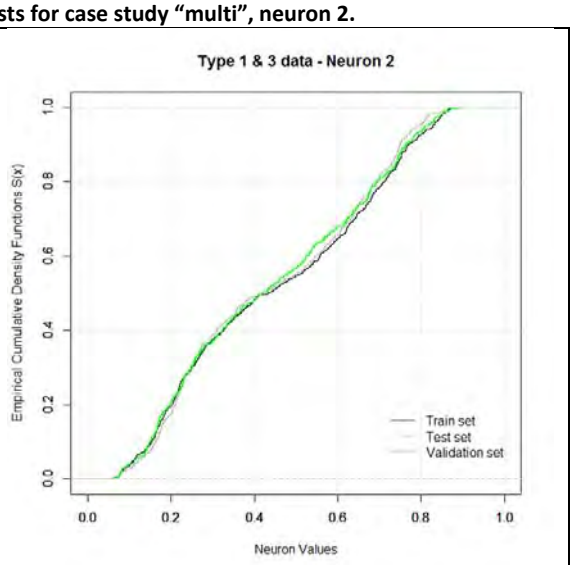


Type 1&3 – Neuron 1	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0917	0.8844	0.6121
CvM Test Statistic	0.2736	0.0523	0.1136

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 29).

Table R8/ 30 - Results of statistical tests for case study "multi", neuron 2.

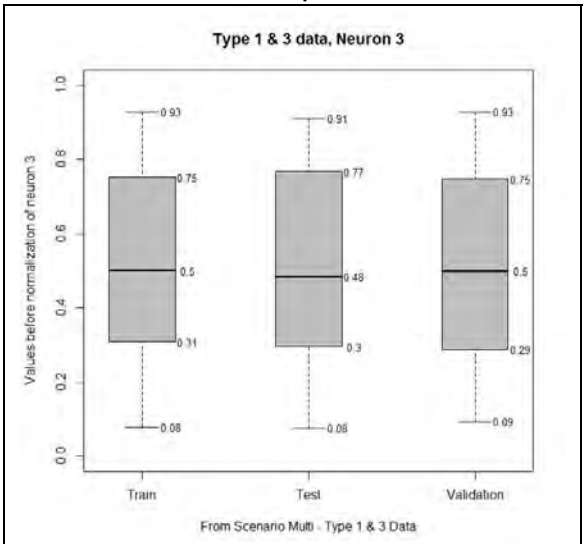


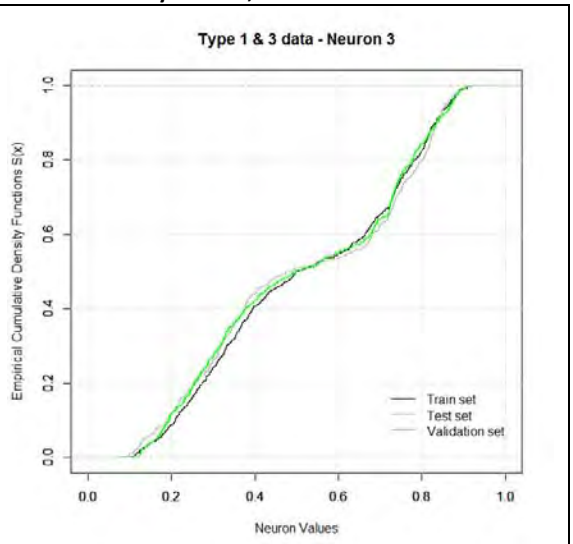


Type 1&3 – Neuron 2	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5387	0.3752	0.7184
CvM Test Statistic	0.1216	0.0851	0.0789

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 30).

Table R8/ 31 - Results of statistical tests for case study "multi", neuron 3.

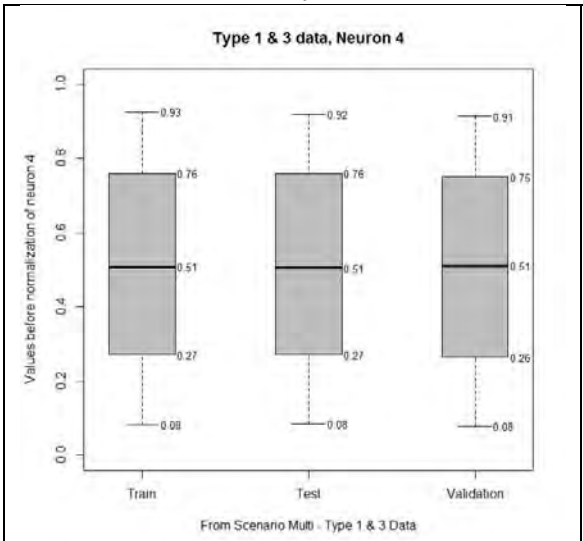


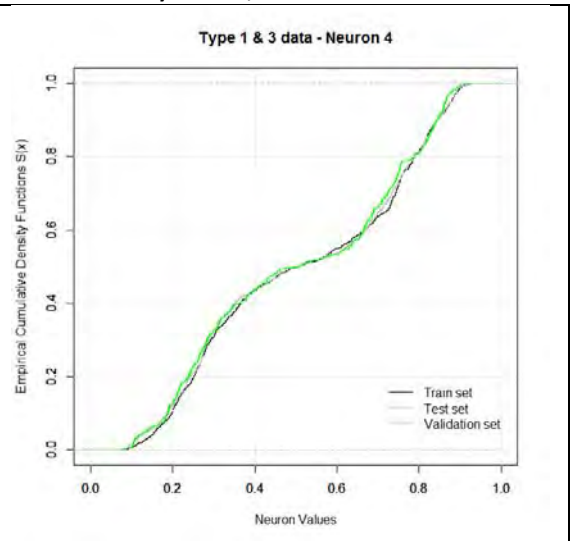


Type 1&3 – Neuron 3	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.3752	0.5095	0.5085
CvM Test Statistic	0.2217	0.1241	0.1029

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 31).

Table R8/ 32 - Results of statistical tests for case study "multi", neuron 4.

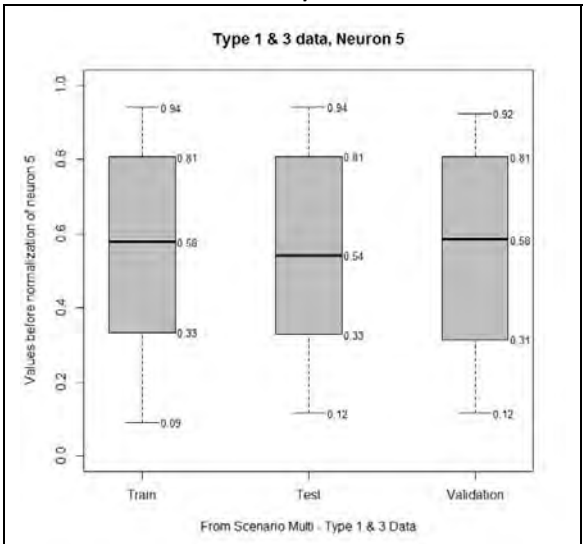


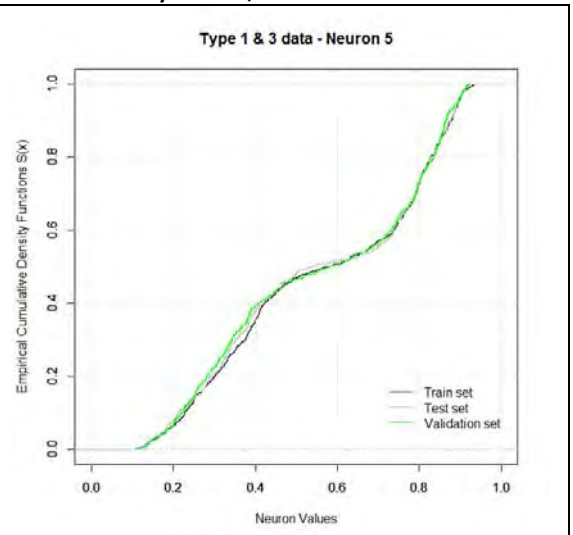


Type 1&3 – Neuron 4	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.9683	0.4002	0.8186
CvM Test Statistic	0.0358	0.1272	0.0462

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 32).

Table R8/ 33 - Results of statistical tests for case study “multi”, neuron 5.

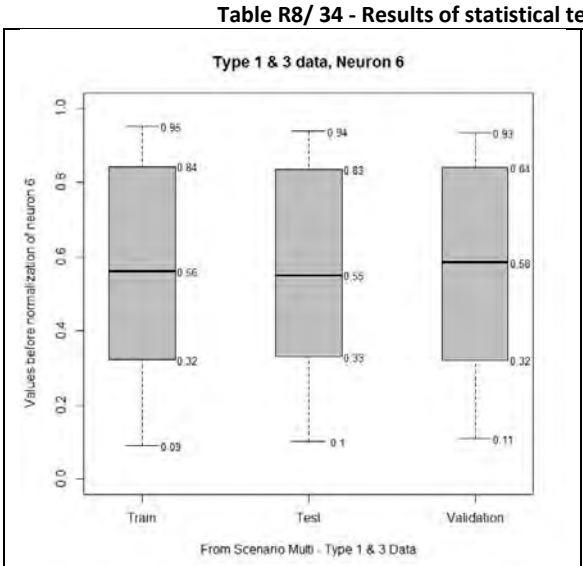


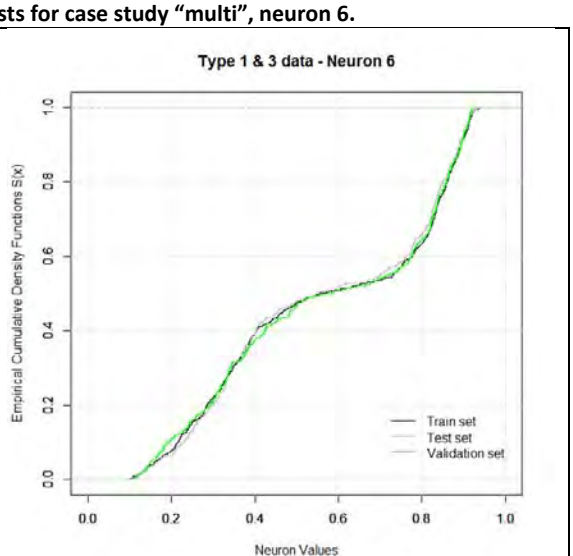


Type 1&3 – Neuron 5	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.9251	0.3512	0.8632
CvM Test Statistic	0.0489	0.1207	0.0460

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 33).

Table R8/ 34 - Results of statistical tests for case study “multi”, neuron 6.

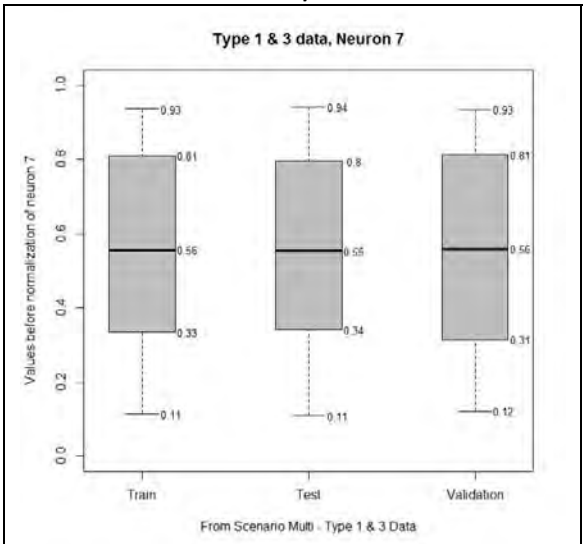


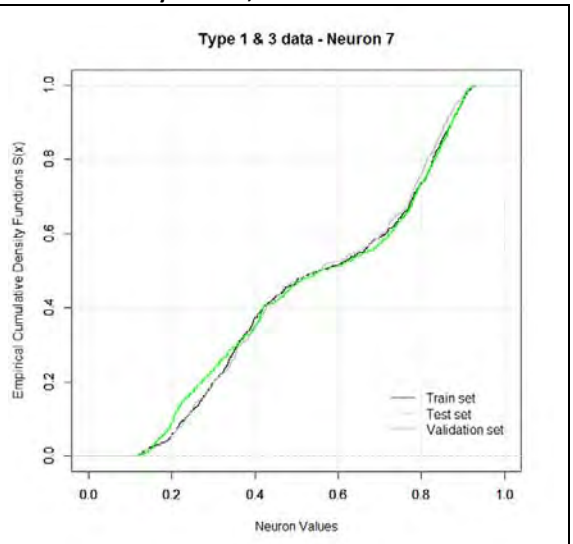


Type 1&3 – Neuron 6	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.7807	0.8844	0.6654
CvM Test Statistic	0.0837	0.0359	0.0743

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 34).

Table R8/ 35 - Results of statistical tests for case study "multi", neuron 7.

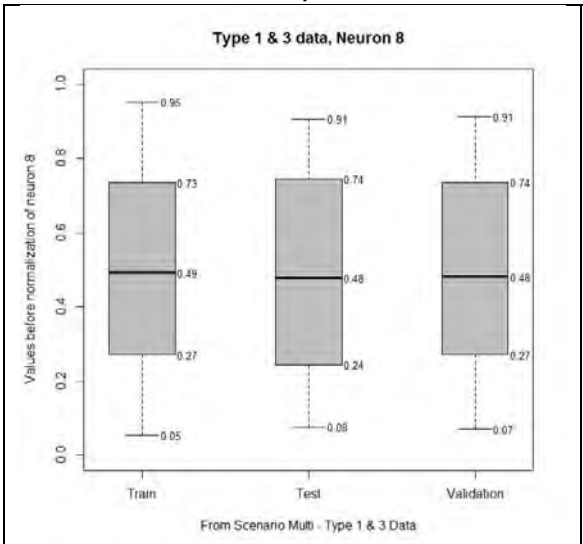


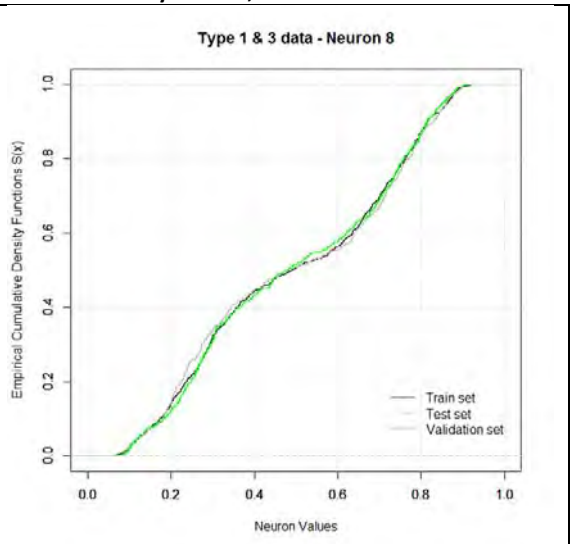


Type 1&3 – Neuron 7	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5686	0.1962	0.2574
CvM Test Statistic	0.0860	0.1526	0.2090

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 35).

Table R8/ 36 - Results of statistical tests for case study "multi", neuron 8.

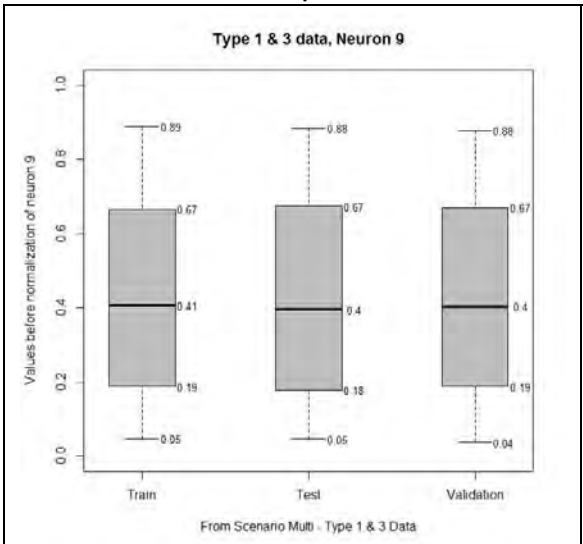


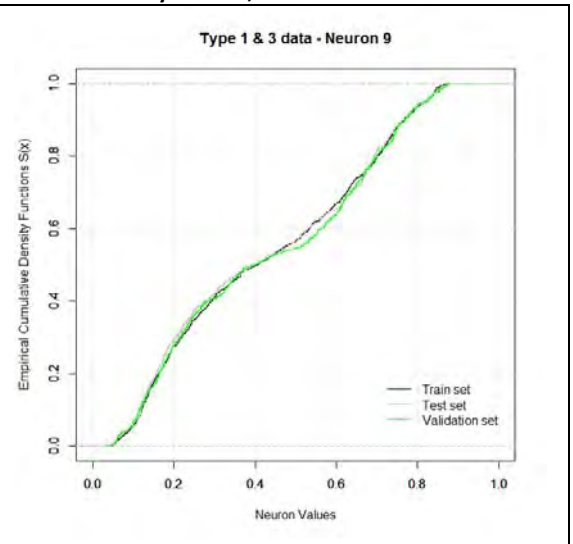


Type 1&3 – Neuron 8	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5686	0.9563	0.3696
CvM Test Statistic	0.0870	0.0354	0.1149

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 36).

Table R8/ 37 - Results of statistical tests for case study "multi", neuron 9.

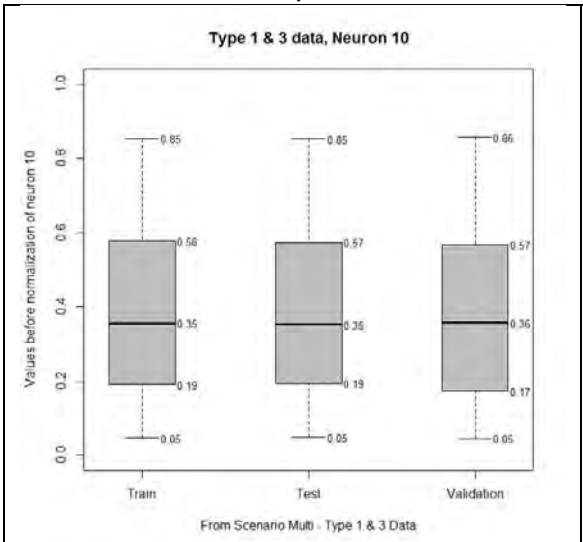


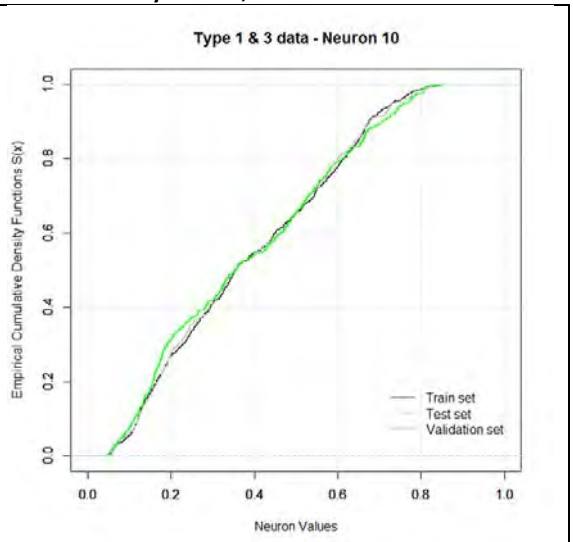


Type 1&3 – Neuron 9	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.8844	0.5990	0.8186
CvM Test Statistic	0.0530	0.0588	0.0511

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 37).

Table R8/ 38 - Results of statistical tests for case study "multi", neuron 10.

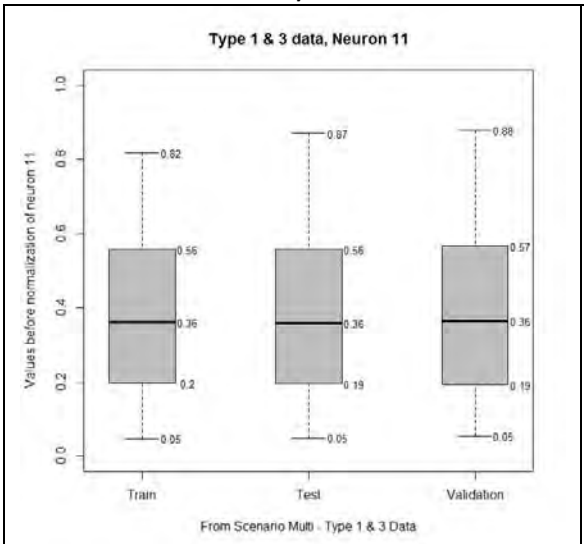


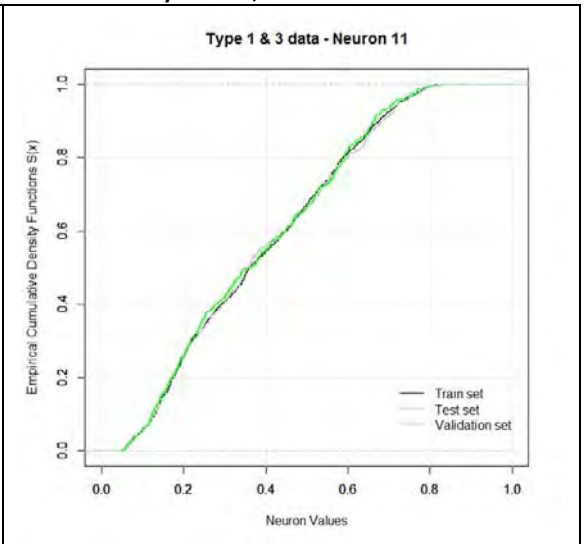


Type 1&3 – Neuron 10	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.8844	0.2656	0.3696
CvM Test Statistic	0.0332	0.1585	0.1020

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 38).

Table R8/ 39 - Results of statistical tests for case study "multi", neuron 11.

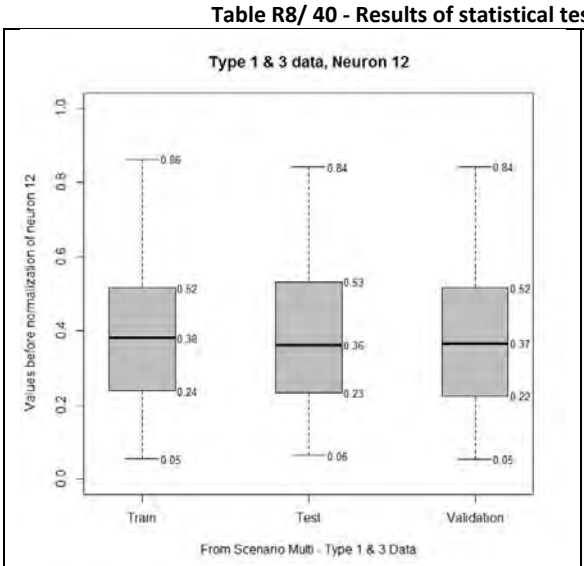


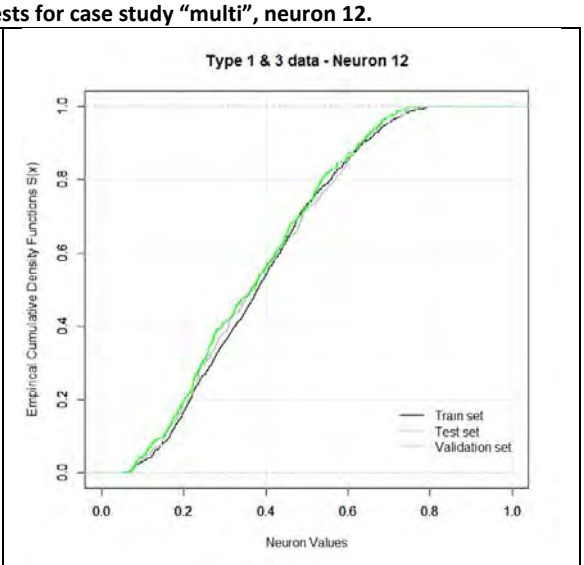


Type 1&3 – Neuron 11	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.9945	0.9251	0.9347
CvM Test Statistic	0.0240	0.0404	0.0493

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 39).

Table R8/ 40 - Results of statistical tests for case study "multi", neuron 12.

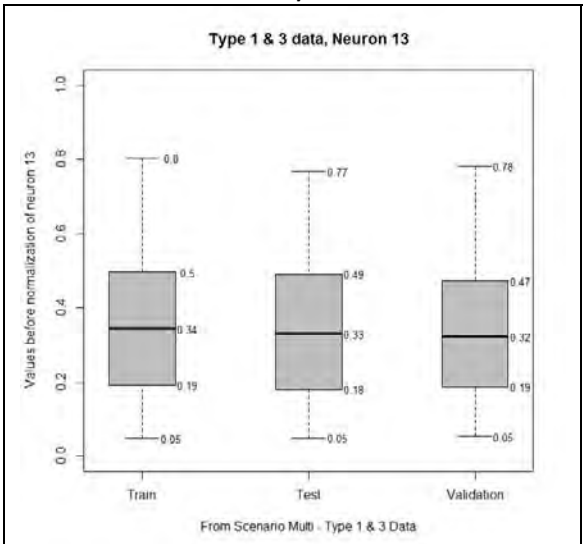


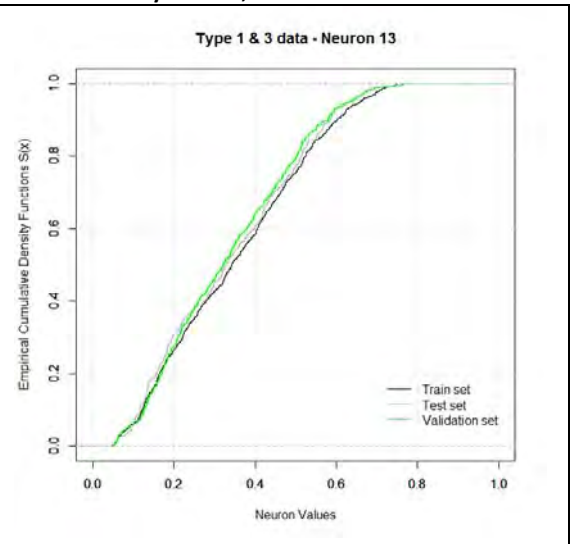


Type 1&3 – Neuron 12	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.6296	0.1096	0.6654
CvM Test Statistic	0.1234	0.2504	0.0868

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 40).

Table R8/ 41 - Results of statistical tests for case study "multi", neuron 13.

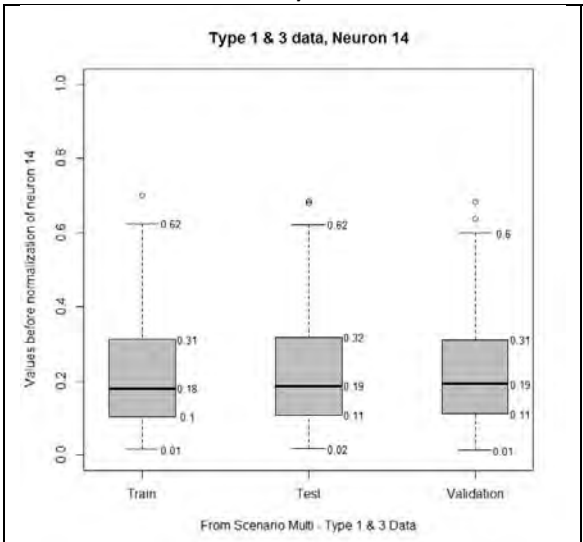


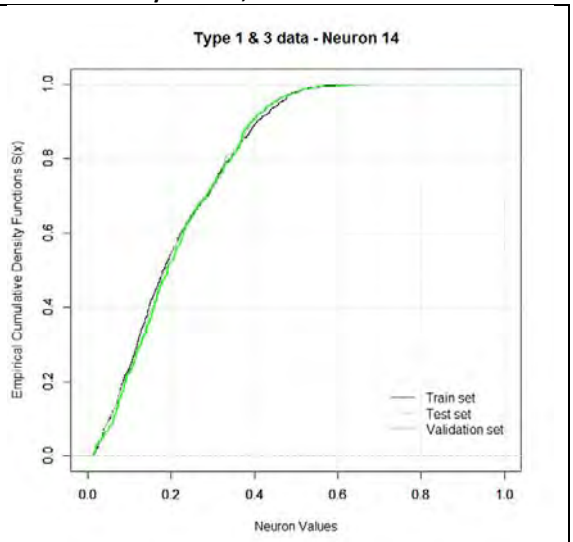


Type 1&3 – Neuron 13	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.4531	0.2121	0.7699
CvM Test Statistic	0.1526	0.3094	0.1170

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 41).

Table R8/ 42 - Results of statistical tests for case study "multi", neuron 14.

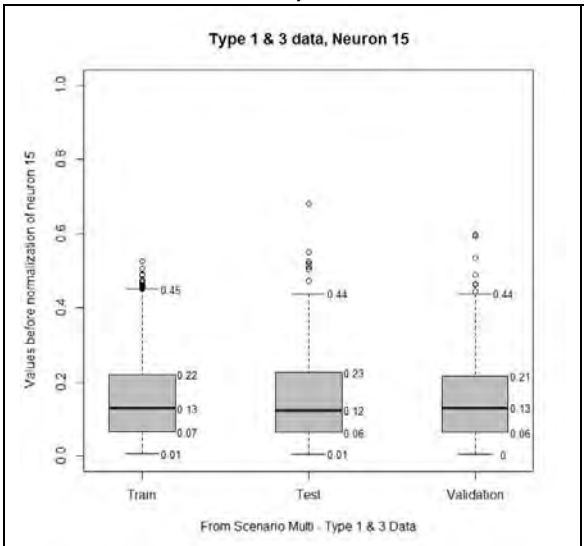


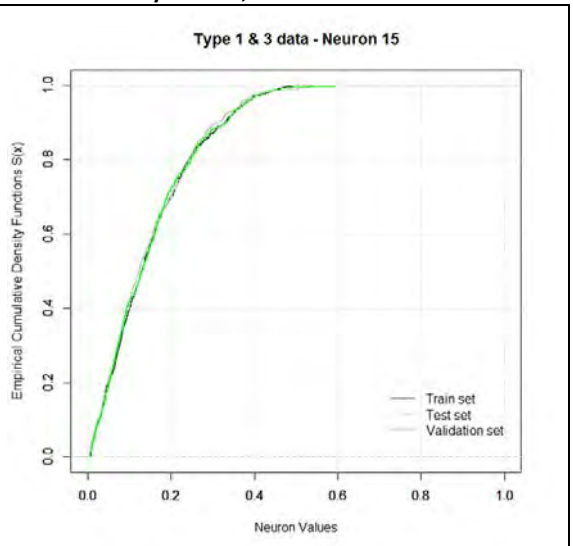


Type 1&3 – Neuron 14	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.7807	0.3512	0.9347
CvM Test Statistic	0.0645	0.1582	0.0390

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 42).

Table R8/ 43 - Results of statistical tests for case study "multi", neuron 15.

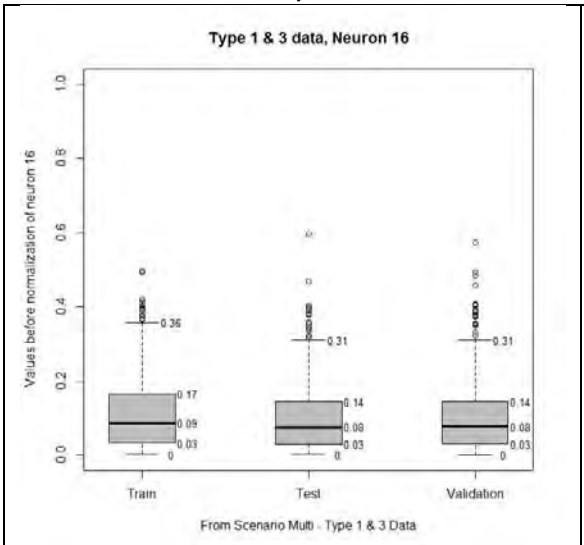


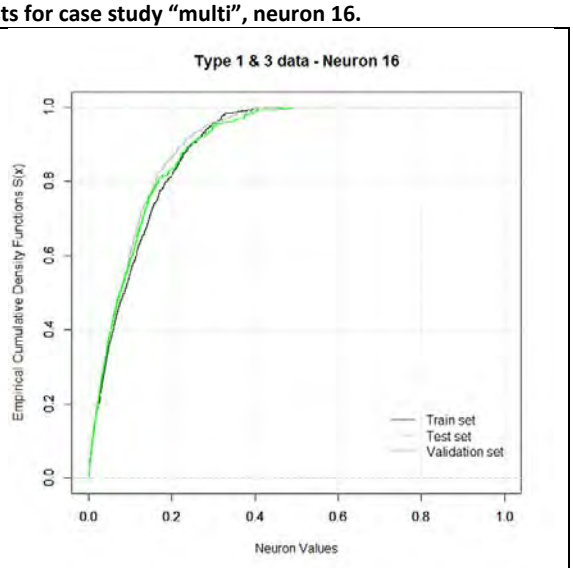


Type 1&3 – Neuron 15	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.9419	0.9779	0.9600
CvM Test Statistic	0.0511	0.0380	0.0536

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 43).

Table R8/ 44 - Results of statistical tests for case study "multi", neuron 16.

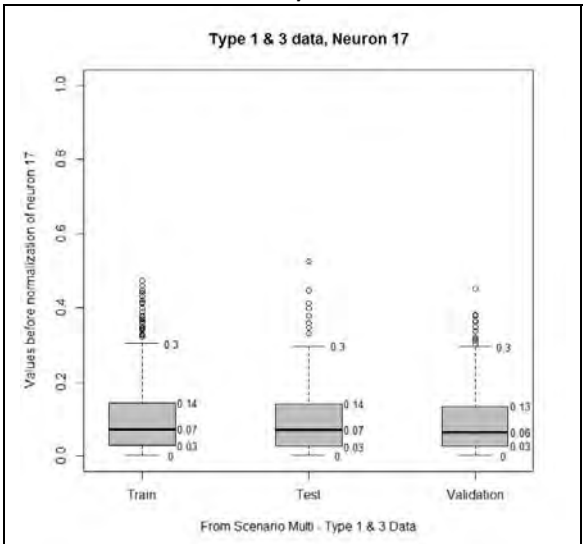


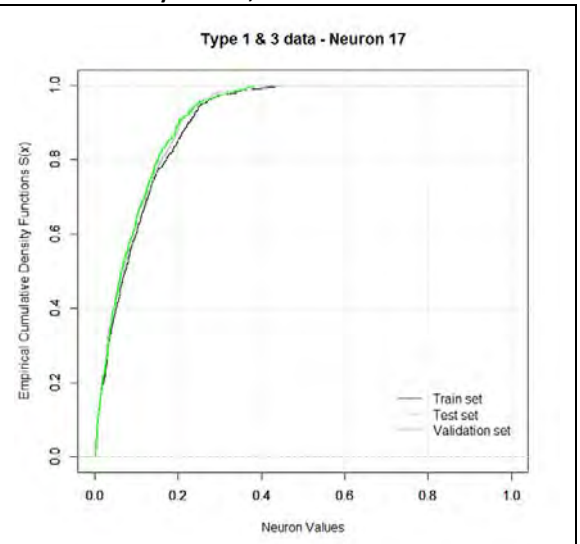


Type 1&3 – Neuron 16	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0346	0.1962	0.8186
CvM Test Statistic	0.4327	0.2197	0.0649

The **Smirnov** test returned the rejection of the null hypothesis for the case Train-Test, with a significance level of 5%. For all the remainder cases, both tests returned the acceptance of the null hypothesis for the same level of significance (see Table R8/ 44).

Table R8/ 45 - Results of statistical tests for case study "multi", neuron 17.

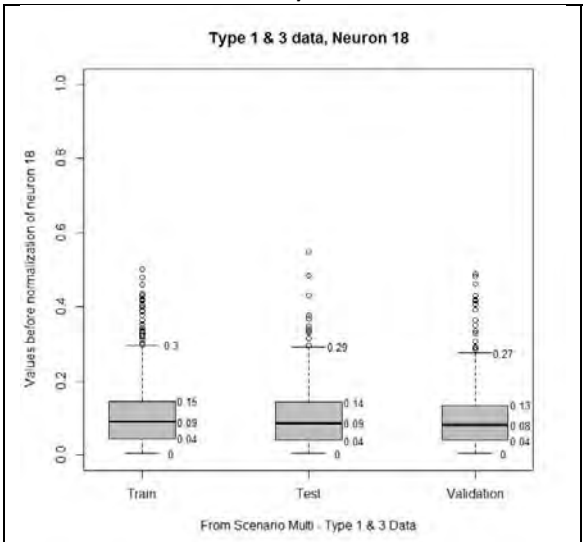


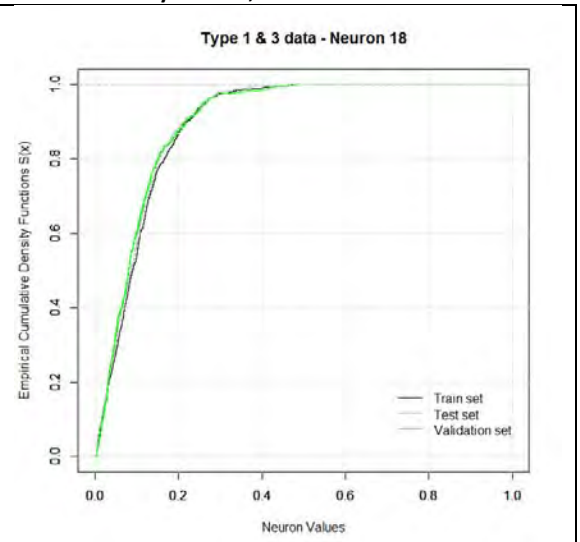


Type 1&3 – Neuron 17	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5686	0.4531	0.8186
CvM Test Statistic	0.1594	0.2018	0.0548

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 45).

Table R8/ 46 - Results of statistical tests for case study "multi", neuron 18.

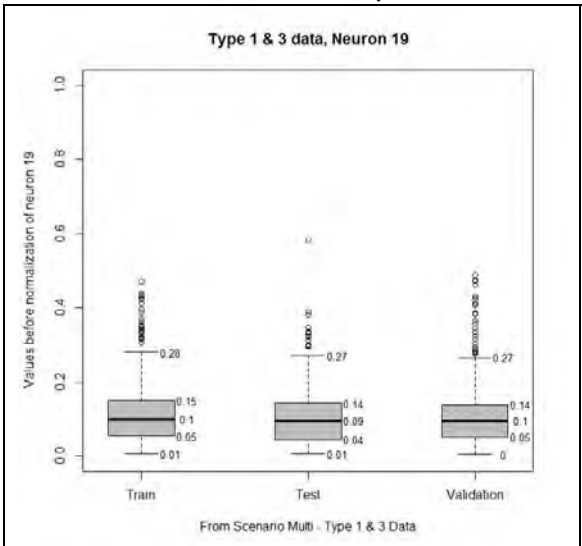


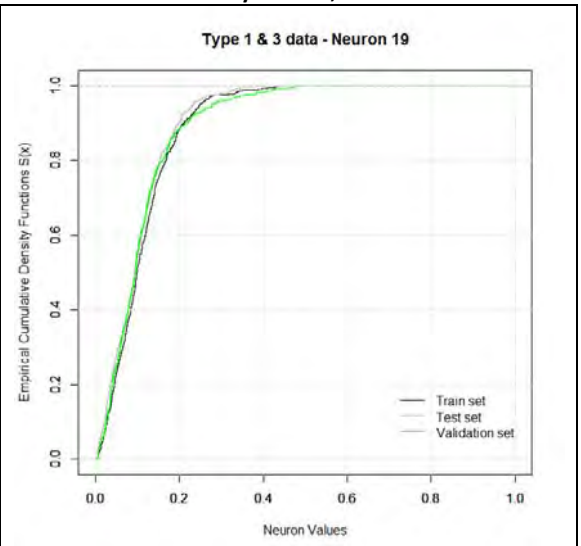


Type 1&3 – Neuron 18	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.5387	0.0837	0.6121
CvM Test Statistic	0.1413	0.4085	0.1175

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 46).

Table R8/ 47 - Results of statistical tests for case study “multi”, neuron 19.

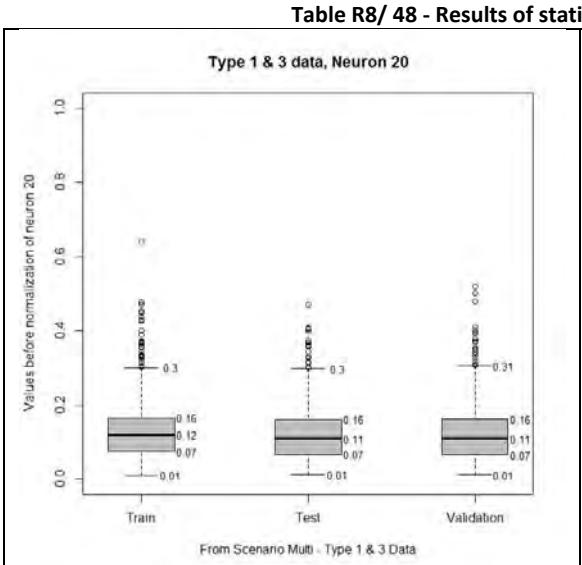


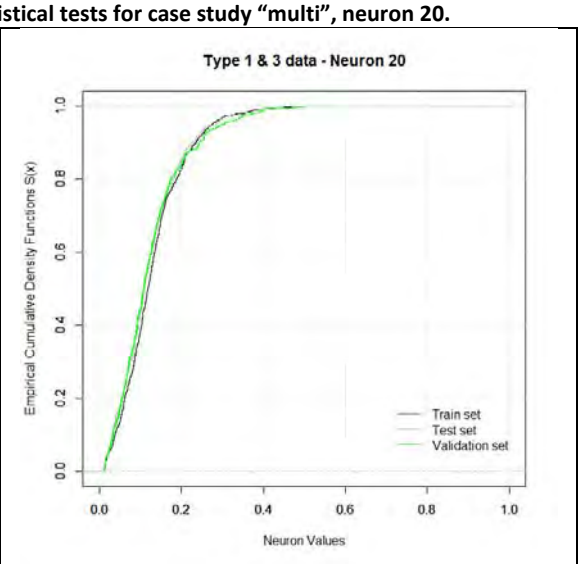


Type 1&3 – Neuron 19	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.1673	0.2289	0.5596
CvM Test Statistic	0.4030	0.3048	0.0954

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 47).

Table R8/ 48 - Results of statistical tests for case study “multi”, neuron 20.

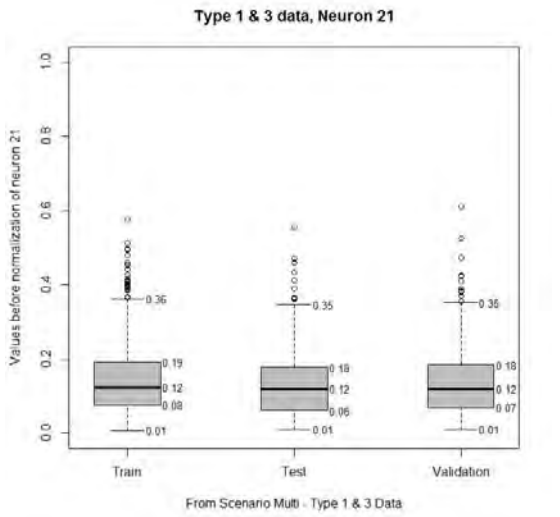


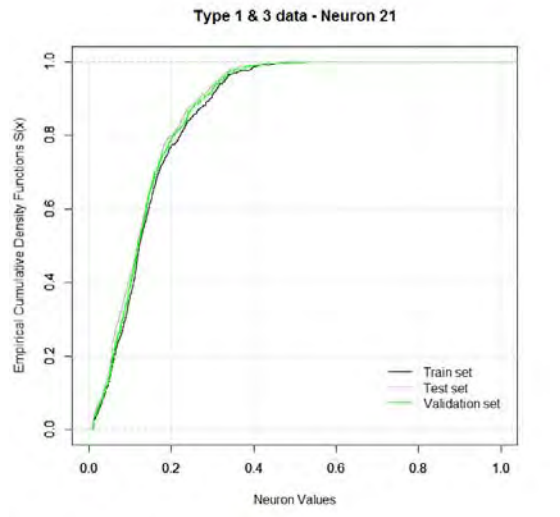


Type 1&3 – Neuron 20	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.1003	0.0425	0.7699
CvM Test Statistic	0.3608	0.5032	0.0431

Both tests replied with the rejection of the null hypothesis for the case Train-Validation, and with the acceptance of the null hypothesis for the cases Train-Test and Test-Validation, considering a significance level of 5% (see Table R8/ 48).

Table R8/ 49 - Results of statistical tests for case study "multi", neuron 21.

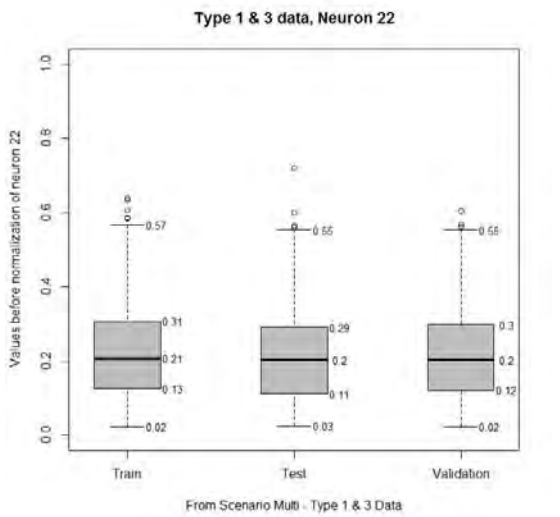


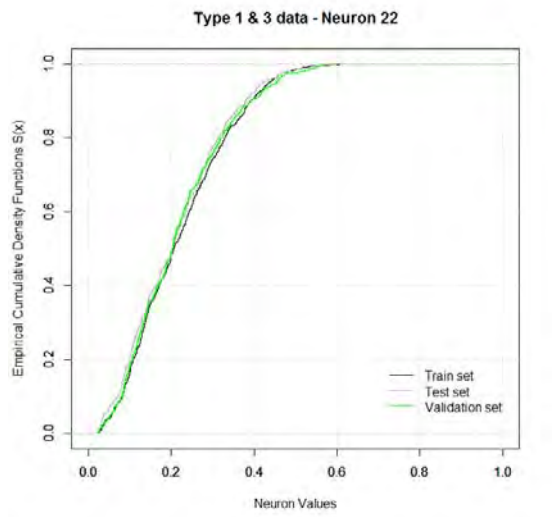


Type 1&3 – Neuron 21	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0763	0.7216	0.5085
CvM Test Statistic	0.4324	0.1318	0.1130

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 49).

Table R8/ 50 - Results of statistical tests for case study "multi", neuron 22.

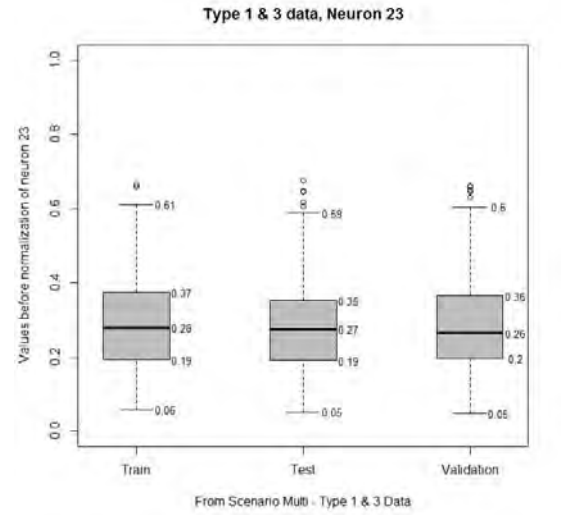


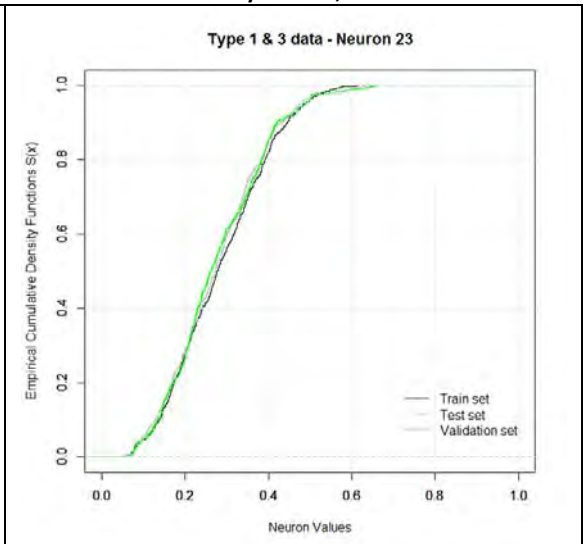


Type 1&3 – Neuron 22	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.3063	0.3752	0.7699
CvM Test Statistic	0.2365	0.1127	0.0880

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 50).

Table R8/ 51 - Results of statistical tests for case study "multi", neuron 23.

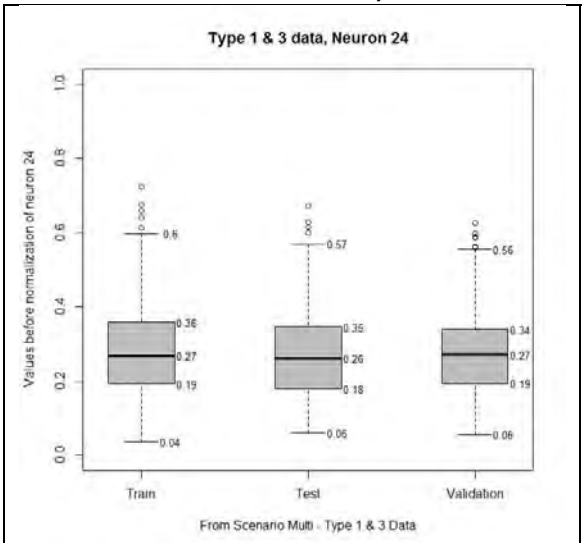


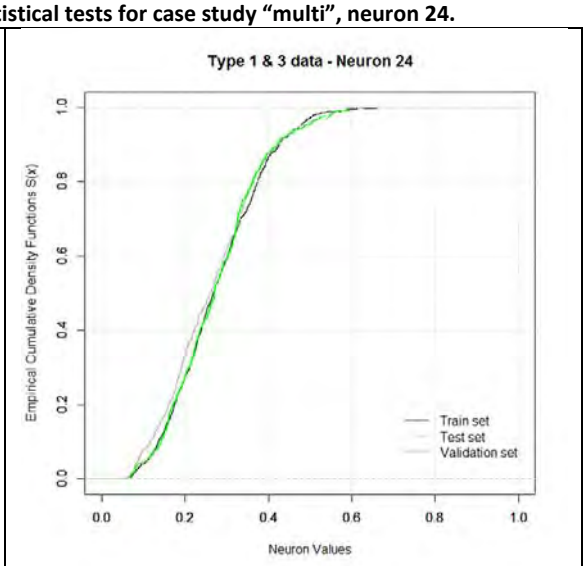


Type 1&3 – Neuron 23	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.4262	0.1541	0.8186
CvM Test Statistic	0.1617	0.2143	0.0714

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 51).

Table R8/ 52 - Results of statistical tests for case study "multi", neuron 24.





Type 1&3 – Neuron 24	Train-Test	Train-Valid	Test-Valid
Smirnov p-value	0.0573	0.5387	0.1979
CvM Test Statistic	0.3706	0.0804	0.2756

The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 52).

82 4 Topology data (Case study "*power system*")

The data used under the "Topology problem" relates to 18 neurons. For each state of the system (Open and Close), three data sets are considered: train, test and validation. The number of examples composing each data set is summarized in Table R8/ 53.

Table R8/ 53 - Number of exemples composing each data set.

Data set	Open System	Closed System
Train	4000	4000
Test	956	1044
Validation	5067	4933

The train and test datasets are used to perform the training of an autoencoder (the evolution of epochs is made using train dataset, and the overfitting assessment is made using the test dataset). This procedure is performed following a "*holdout*" approach (or split sample test).

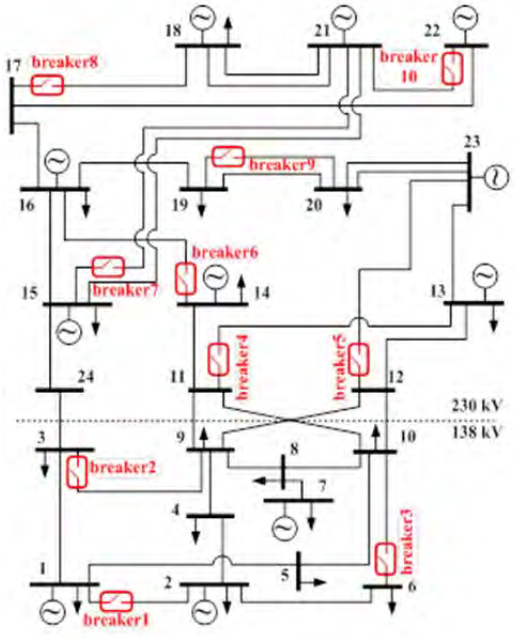


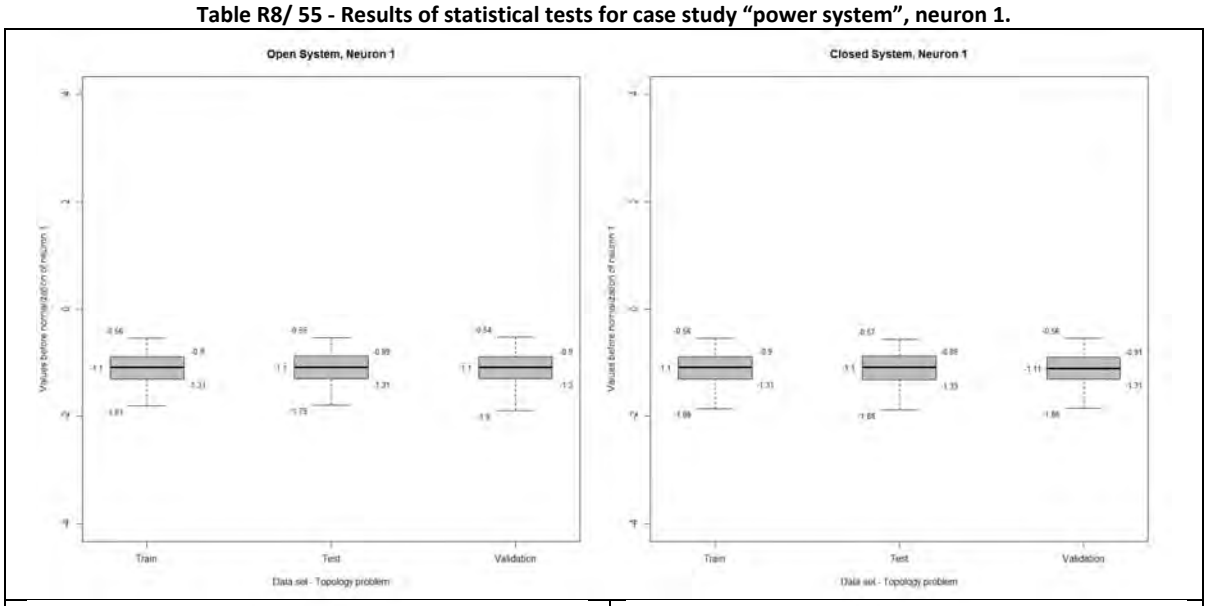
Figure R8/ 1 - IEEE RTS 24 (IEEE RTS Task Force of APM Subcommittee, 1979) with identification of breakers' position.

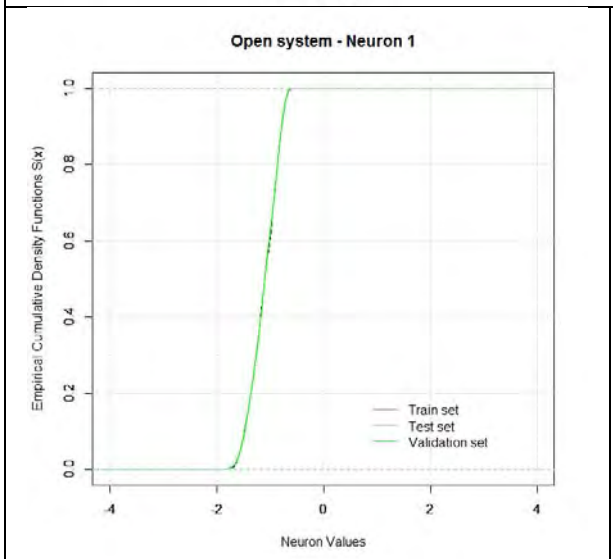
The validation dataset is used to calculate the number of fails, knowing that this dataset is independent from the training process. The results concerning the application of statistical test is summarized in Table R8/ 54.

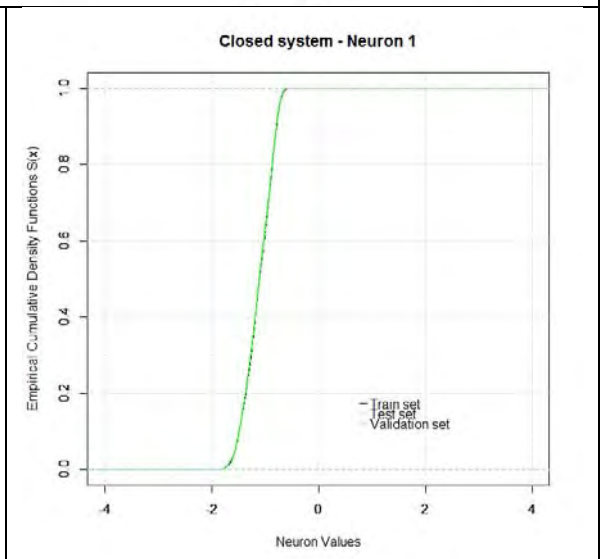
Table R8/ 54 - Summary of the statistic test conducted, "yes" indicates the acceptance of the null hypothesis and "no" the rejection of the null hypothesis (α=5%).

	Open						Closed					
	Train-Test:		Train-Valid:		Test-Valid		Train-Test:		Train-Valid:		Test-Valid	
	Smirnov	CvM	Smirnov	CvM	Smirnov	CvM	Smirnov	CvM	Smirnov	CvM	Smirnov	CvM
1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
2	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes	yes
3	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
4	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
5	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
6	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
7	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
8	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
9	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
10	yes	yes	no	no	yes	yes	yes	yes	yes	yes	yes	yes
11	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes	yes
12	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes	yes
13	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
14	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
15	yes	yes	no	no	yes	yes	yes	yes	yes	yes	yes	yes
16	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
17	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
18	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

Table R8/ 55 - Results of statistical tests for case study "power system", neuron 1.



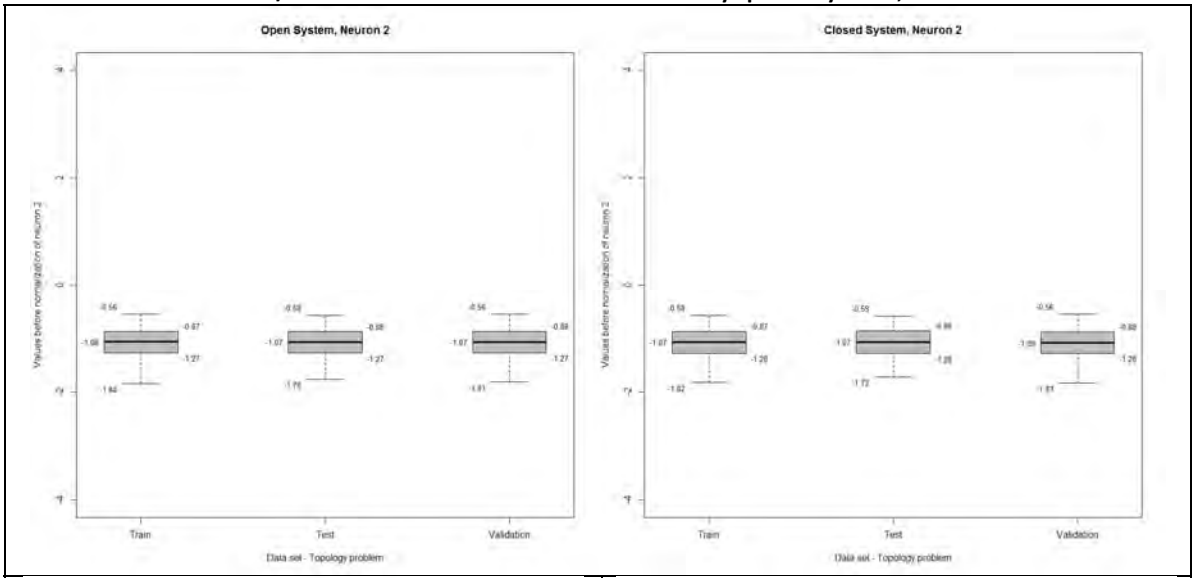


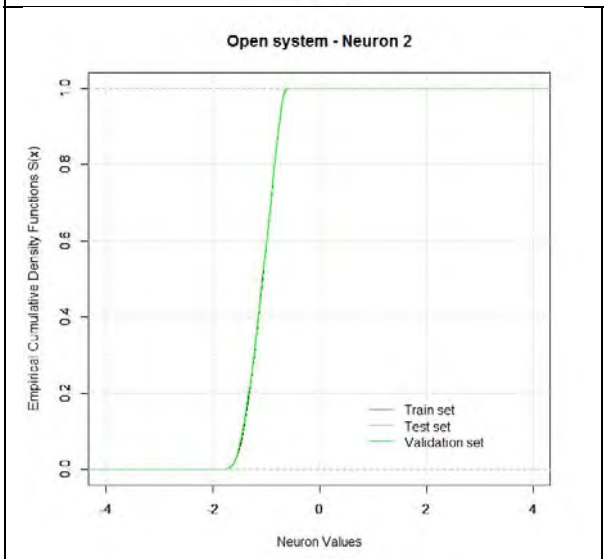


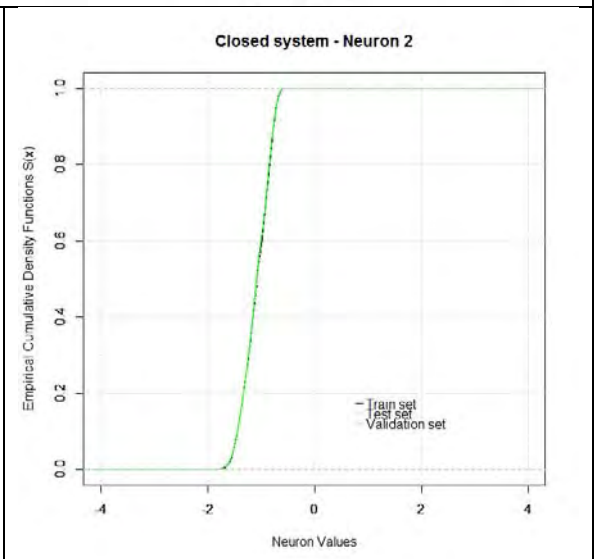
Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.8383	0.0498	Train - Test	0.7454	0.0618
Train-Valid	0.5493	0.0722	Train-Valid	0.0552	0.3695
Test - Valid	0.9539	0.0475	Test - Valid	0.2619	0.2518

Neuron 1 has the label P_inj3 and relates to: active power injection. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 55).

Table R8/ 56 - Results of statistical tests for case study "power system", neuron 2.



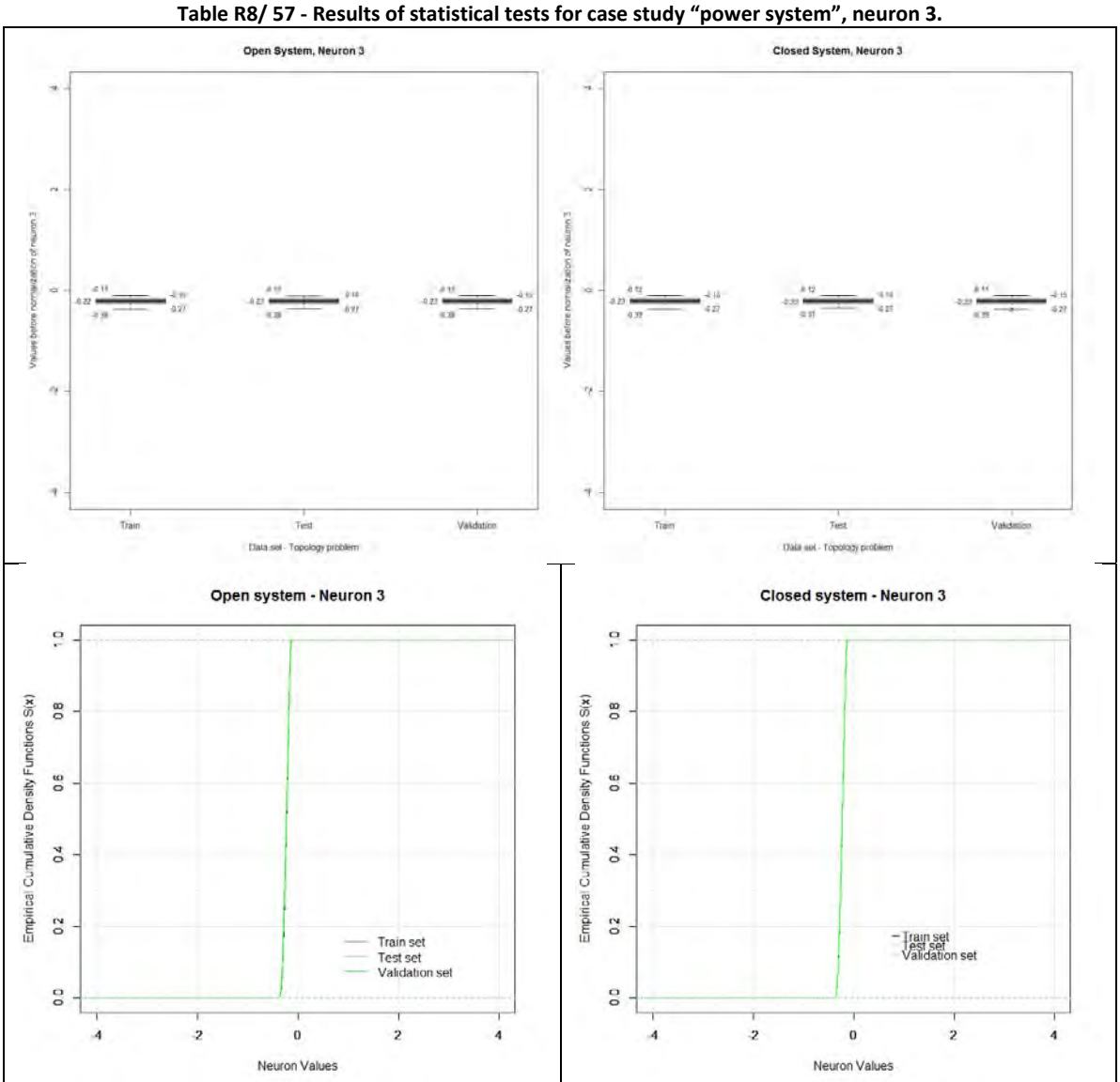




Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.9772	0.0456	Train - Test	0.4139	0.1017
Train-Valid	0.7114	0.0551	Train-Valid	0.0148	0.5819
Test - Valid	0.9703	0.0329	Test - Valid	0.1437	0.4290

Neuron 2 has the label P_inj9 and relates to active power injection. Both tests replied with the rejection of the null hypothesis for the case Train Validation within the closed system, consequently the similarity amid these data sets is not assumed. Both tests returned with the acceptance of the null hypothesis for all the remainder cases, which indicates the population similarity, always considering a significance level of 5% (see Table R8/ 56).

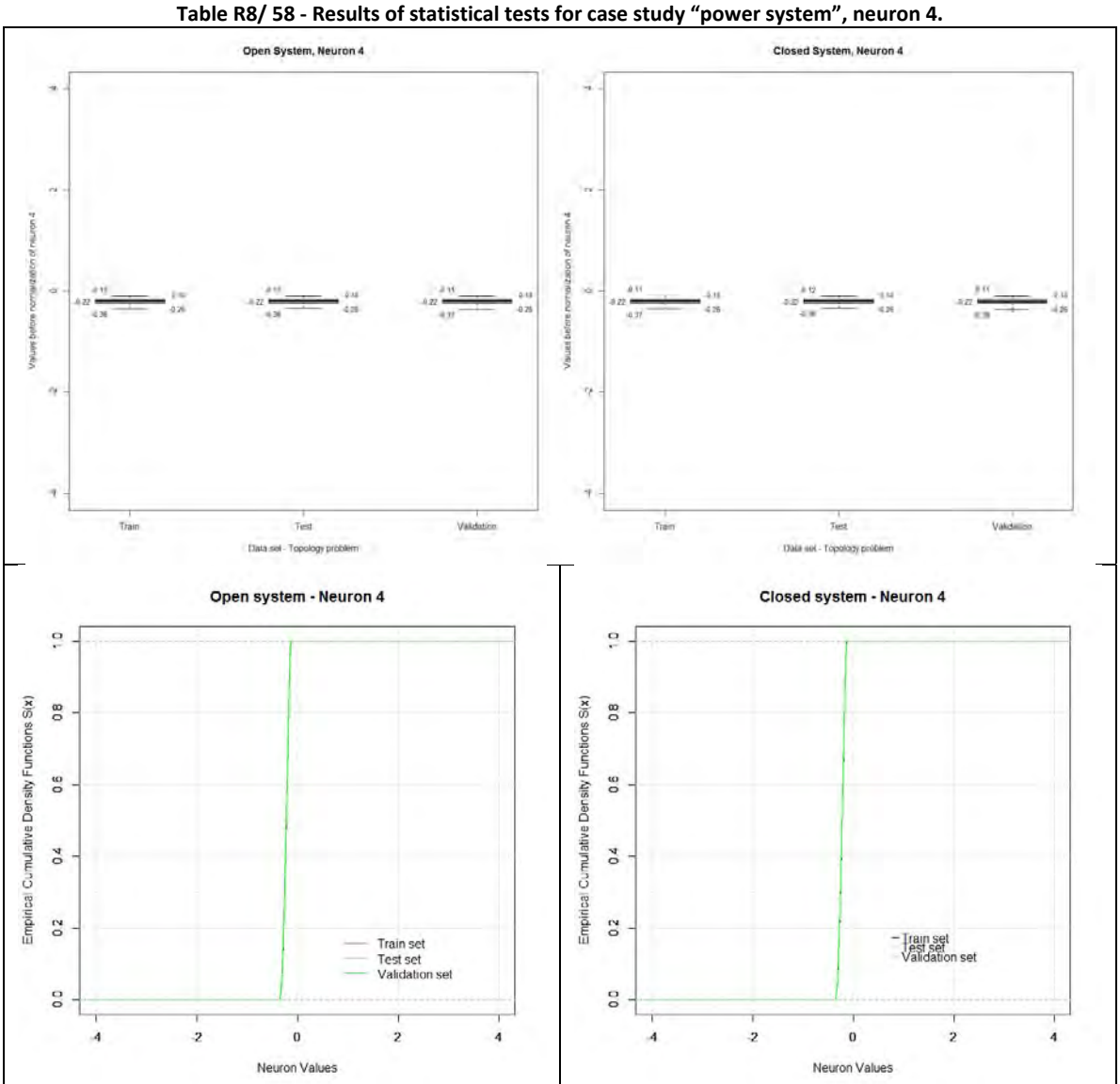
Table R8/ 57 - Results of statistical tests for case study "power system", neuron 3.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.8827	0.0814	Train - Test	0.9017	0.0339
Train-Valid	0.8102	0.0979	Train-Valid	0.1089	0.3895
Test - Valid	0.8337	0.0515	Test - Valid	0.3744	0.2417

Neuron 3 has the label Q_inj3 and relates to reactive power injection. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 57).

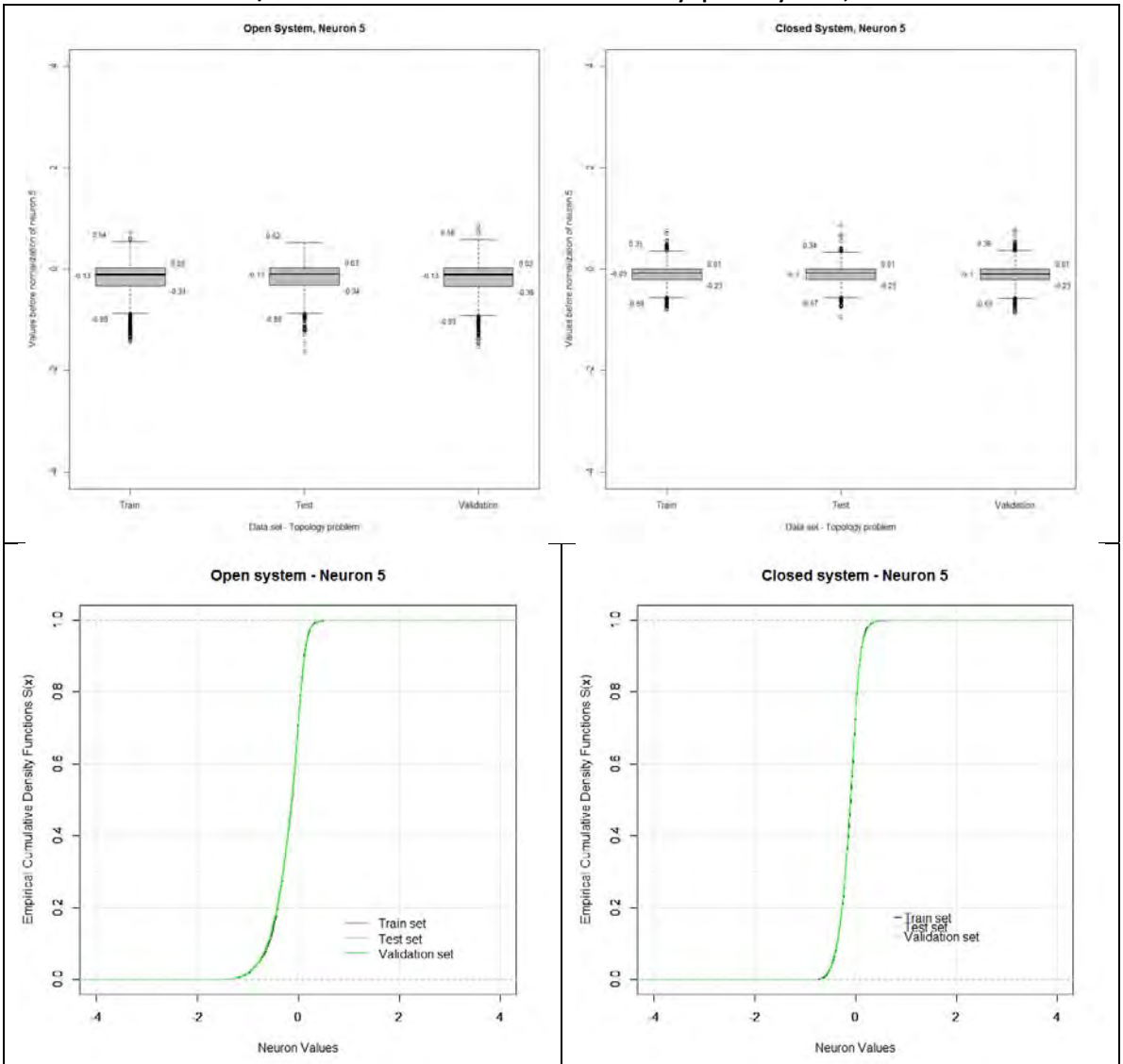
Table R8/ 58 - Results of statistical tests for case study “power system”, neuron 4.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.8392	0.0425	Train - Test	0.5153	0.1312
Train-Valid	0.7193	0.0487	Train-Valid	0.1452	0.3277
Test - Valid	0.8736	0.0632	Test - Valid	0.1266	0.3076

Neuron 4 has the label Q_inj9 and relates to reactive power injection. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 58).

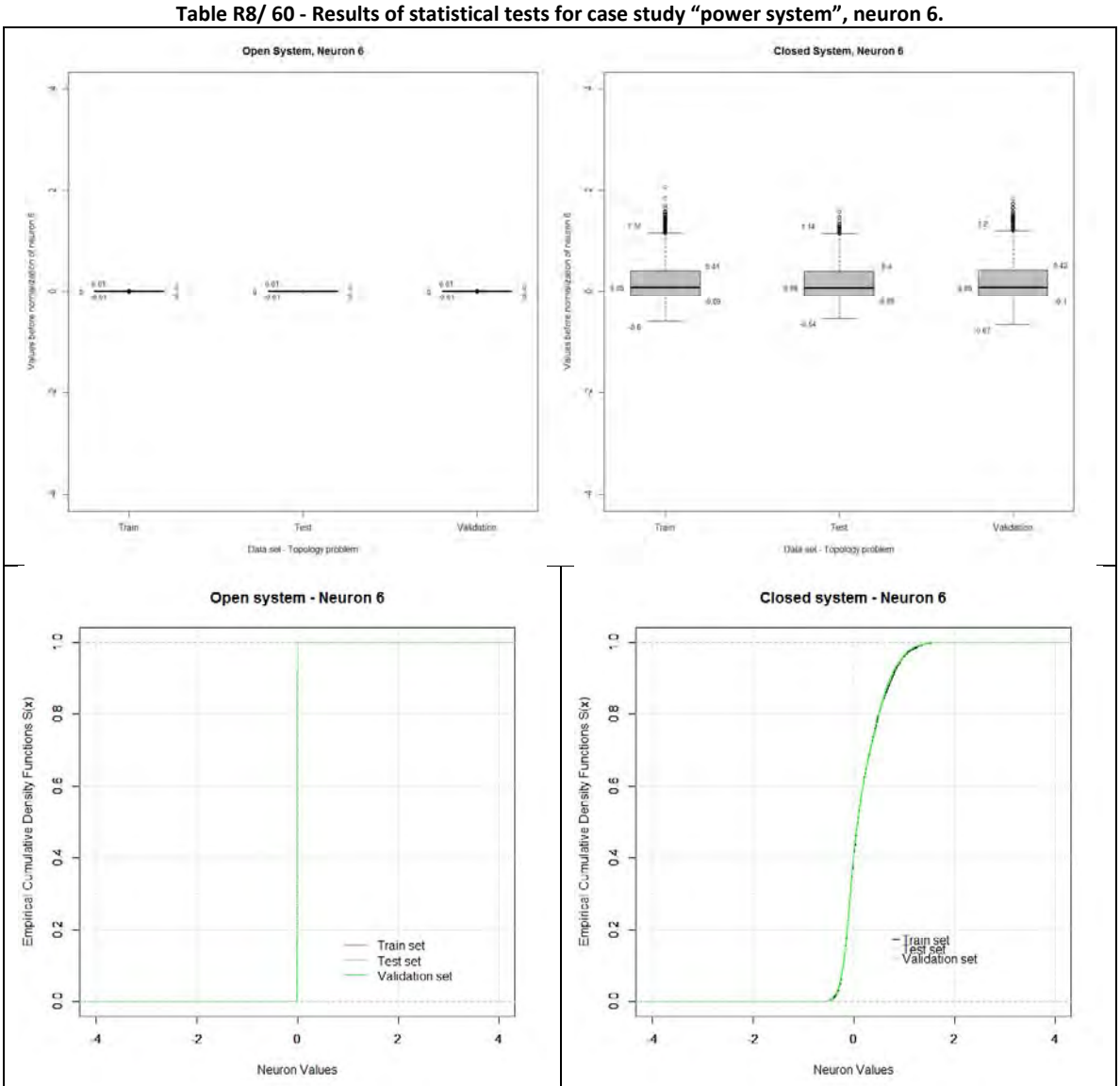
Table R8/ 59 - Results of statistical tests for case study "power system", neuron 5.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.1096	0.2203	Train - Test	0.9321	0.0511
Train-Valid	0.4094	0.1625	Train-Valid	0.0682	0.2951
Test - Valid	0.0860	0.2716	Test - Valid	0.8170	0.0696

Neuron 5 has the label P_flow1-3 and relates to active power flow on the line 1-3. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 59).

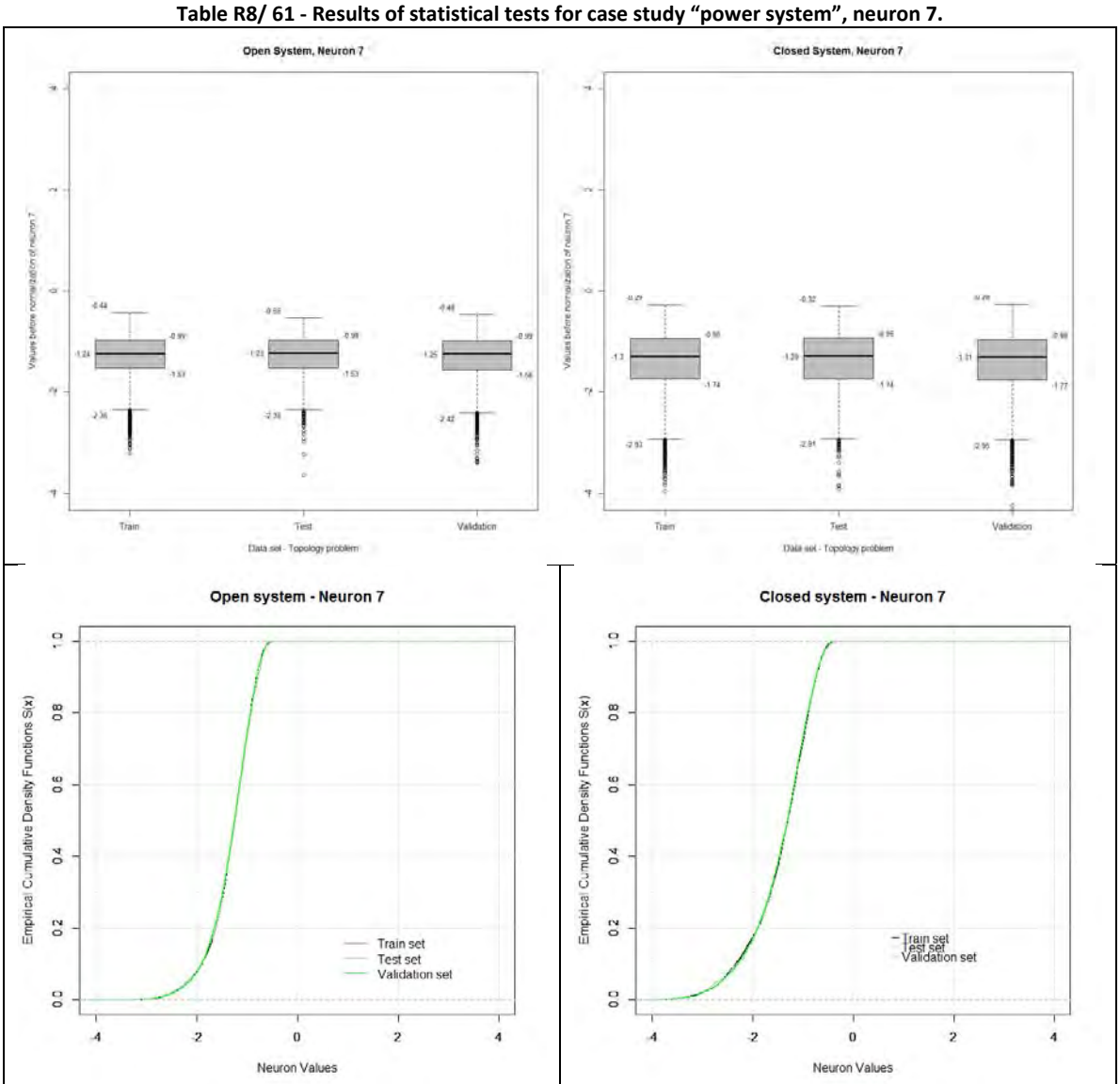
Table R8/ 60 - Results of statistical tests for case study "power system", neuron 6.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.0078	0.6562	Train - Test	0.9580	0.0327
Train-Valid	0.2079	0.2642	Train-Valid	0.7654	0.0778
Test - Valid	0.0755	0.3478	Test - Valid	0.7413	0.0463

Neuron 6 has the label P_flow3-9 and relates to: active power flow on the line 3-9. Both tests replied with the rejection of the null hypothesis for the case Train Validation within the open system, indicating the non-similarity between these two data sets. Both tests returned the acceptance of the null hypothesis for all the remainder cases, indicating the similarity among each pair of data sets, always considering a significance level of 5% (see Table R8/ 60).

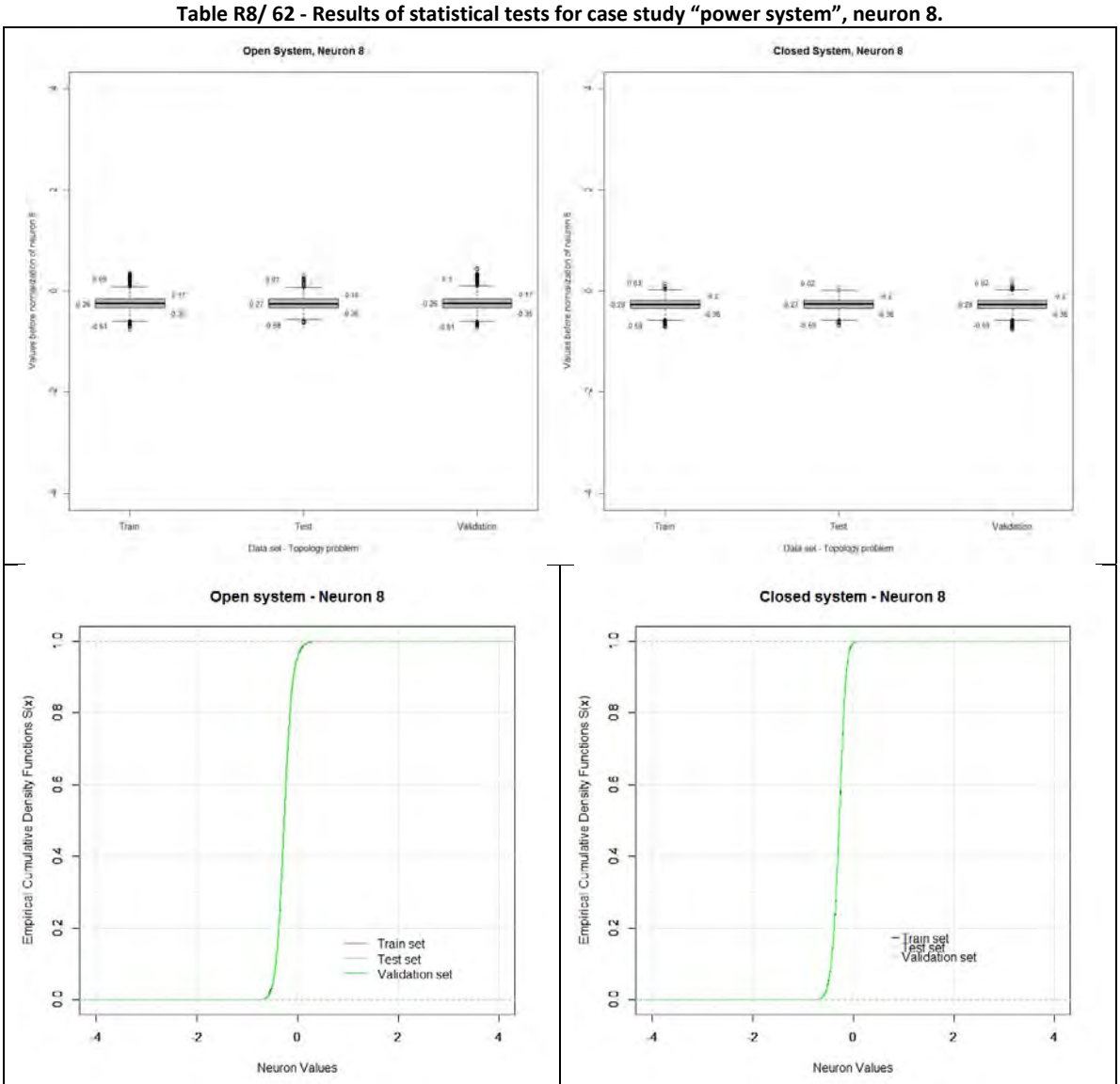
Table R8/ 61 - Results of statistical tests for case study "power system", neuron 7.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.4144	0.0795	Train - Test	0.9787	0.0356
Train-Valid	0.2883	0.1887	Train-Valid	0.3175	0.1933
Test - Valid	0.6543	0.1607	Test - Valid	0.4960	0.1642

Neuron 7 has the label P_flow3-24 and relates to active power flow on the line 3-24. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 61).

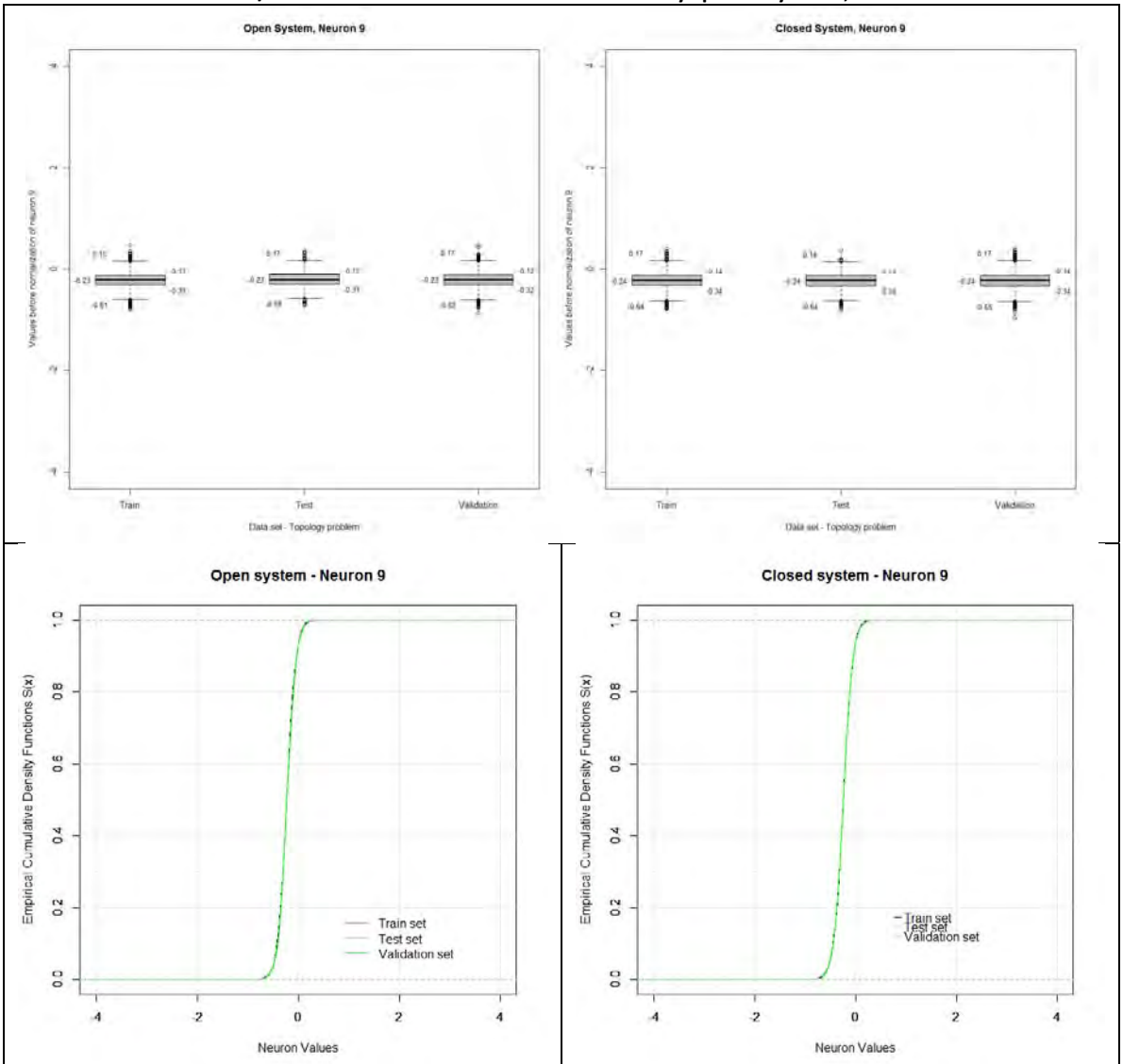
Table R8/ 62 - Results of statistical tests for case study "power system", neuron 8.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.9106	0.0795	Train - Test	0.8266	0.0533
Train-Valid	0.1361	0.2837	Train-Valid	0.5061	0.0972
Test - Valid	0.4265	0.1776	Test - Valid	0.4433	0.1118

Neuron 8 has the label P_flow4-9 and relates to active power flow on the line 4-9. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 62).

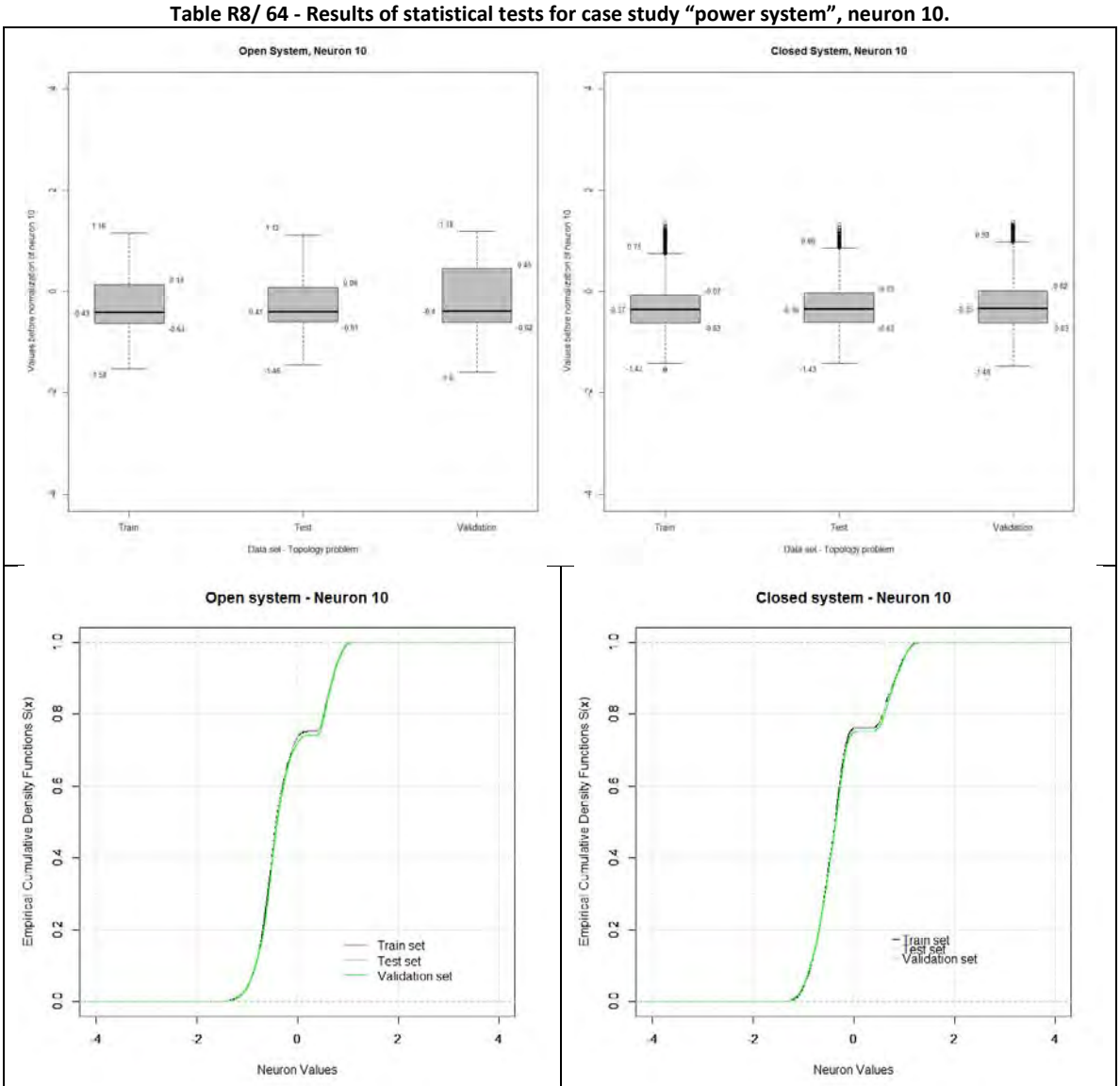
Table R8/ 63 - Results of statistical tests for case study "power system", neuron 9.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.2002	0.3274	Train - Test	0.9868	0.0249
Train-Valid	0.1081	0.2662	Train-Valid	0.6643	0.0880
Test - Valid	0.5995	0.0970	Test - Valid	0.7285	0.0480

Neuron 9 has the label P_flow8-9 and relates to active power flow on the line 8-9. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 63).

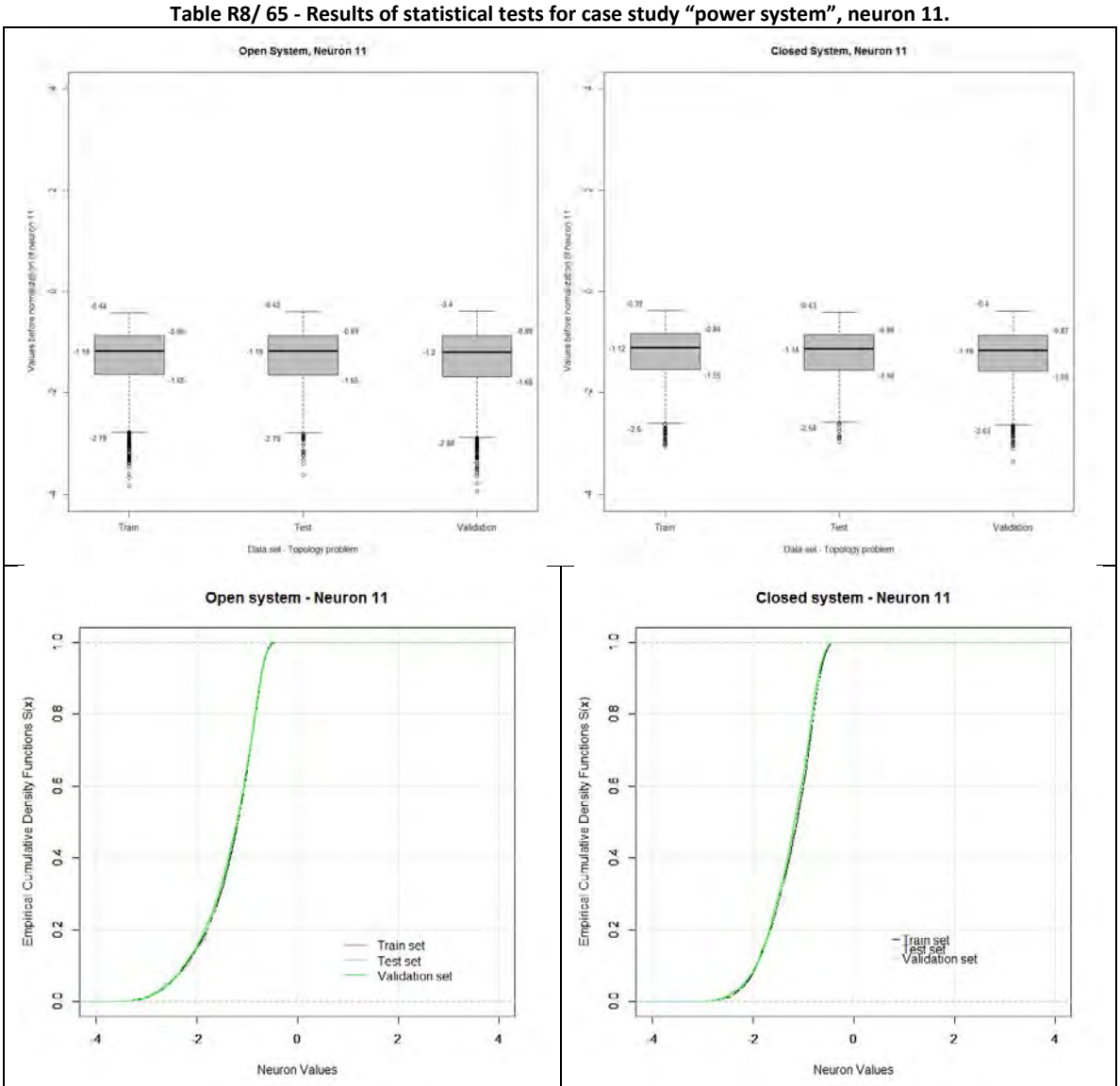
Table R8/ 64 - Results of statistical tests for case study "power system", neuron 10.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.1666	0.1831	Train - Test	0.7491	0.0614
Train-Valid	0.0102	0.6138	Train-Valid	0.0929	0.3631
Test - Valid	0.8445	0.0707	Test - Valid	0.8577	0.0649

Neuron 10 has the label P_flow9-11 and relates to active power flow on the line 9-11. The null hypothesis was rejected for the case Train-Validation within the open system with both tests, indicating the non-similarity of this pair of data sets for a significance level of 5%. The null hypothesis was accepted by both tests for all the remainder cases, indicating the population similarity among each pair of data sets, using a significance level of 5% (see Table R8/ 64).

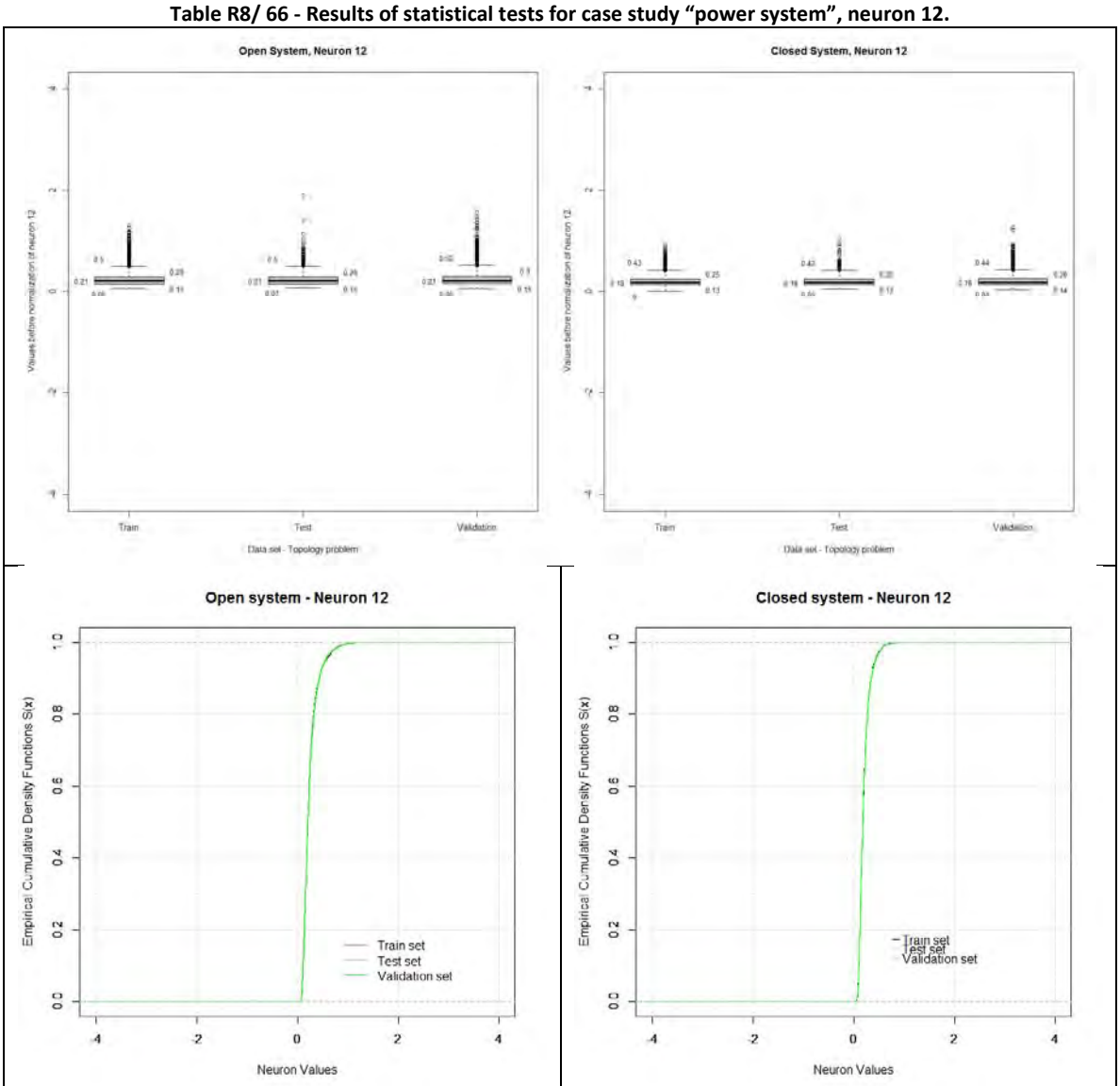
Table R8/ 65 - Results of statistical tests for case study "power system", neuron 11.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.7292	0.0588	Train - Test	0.6393	0.1172
Train-Valid	0.1451	0.2706	Train-Valid	0.0016	1.3402
Test - Valid	0.4713	0.0565	Test - Valid	0.2750	0.1973

Neuron 11 has the label P_flow9-12 relates to active power flow on the line 9-12. The null hypothesis was rejected for the case Train-Validation within the closed system with both tests, indicating the non-similarity of this pair of data sets using a significance level of 5%. The null hypothesis was accepted by both tests for all the remainder cases, indicating the population similarity among each pair of data sets, using a significance level of 5% (see Table R8/ 65).

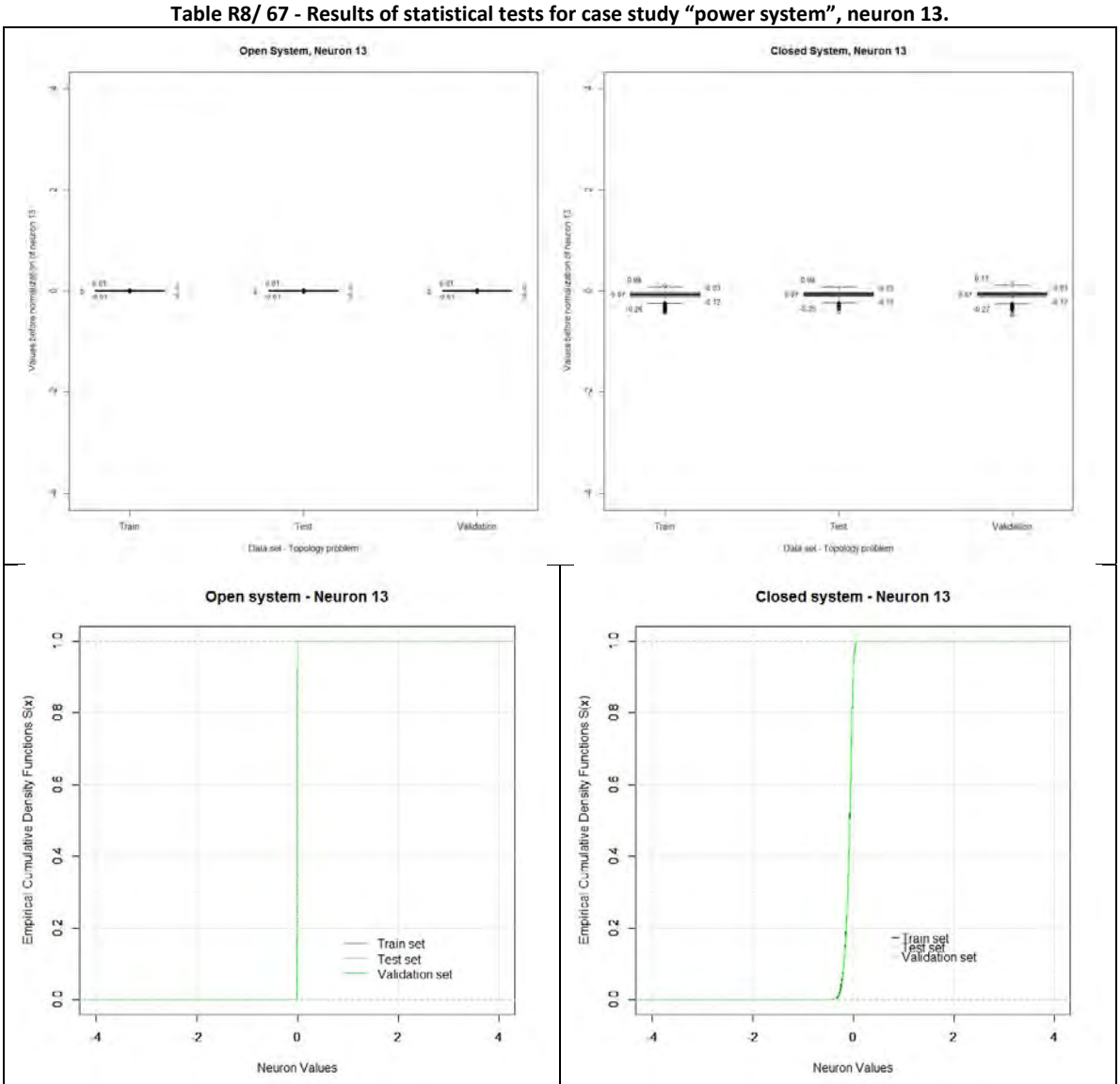
Table R8/ 66 - Results of statistical tests for case study "power system", neuron 12.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.5485	0.1262	Train - Test	0.8731	0.0646
Train-Valid	0.1615	0.2332	Train-Valid	0.0444	0.5250
Test - Valid	0.3226	0.2234	Test - Valid	0.0633	0.3583

Neuron 12 has the label Q_flow1-3 and relates to reactive power flow on the line 1-3. The null hypothesis was rejected for the case Train-Validation within the closed system with both tests, which indicates the non-similarity of this pair of data sets, considering a 5% significance level. For all the remainder cases, the null hypothesis was accepted by both tests, indicating the population similarity among each pair of data sets, also with a 5% significance level (see Table R8/ 66).

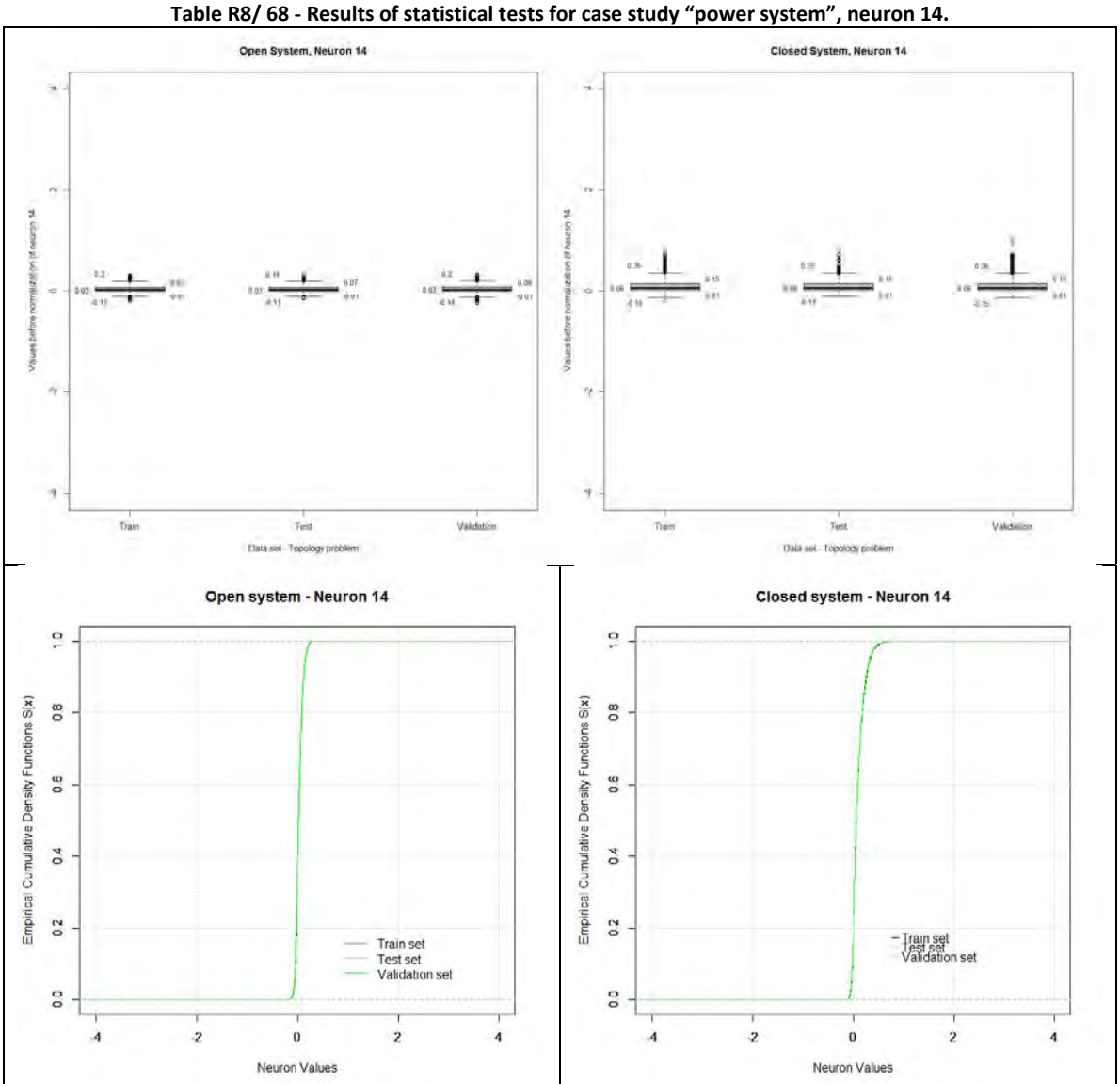
Table R8/ 67 - Results of statistical tests for case study "power system", neuron 13.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.7298	0.0479	Train - Test	0.7440	0.0986
Train-Valid	0.5453	0.1507	Train-Valid	0.5897	0.1064
Test - Valid	0.4723	0.1296	Test - Valid	0.6650	0.1140

Neuron 13 has the label Q_flow3-9 and relates to reactive power flow on the line 3-9. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 67).

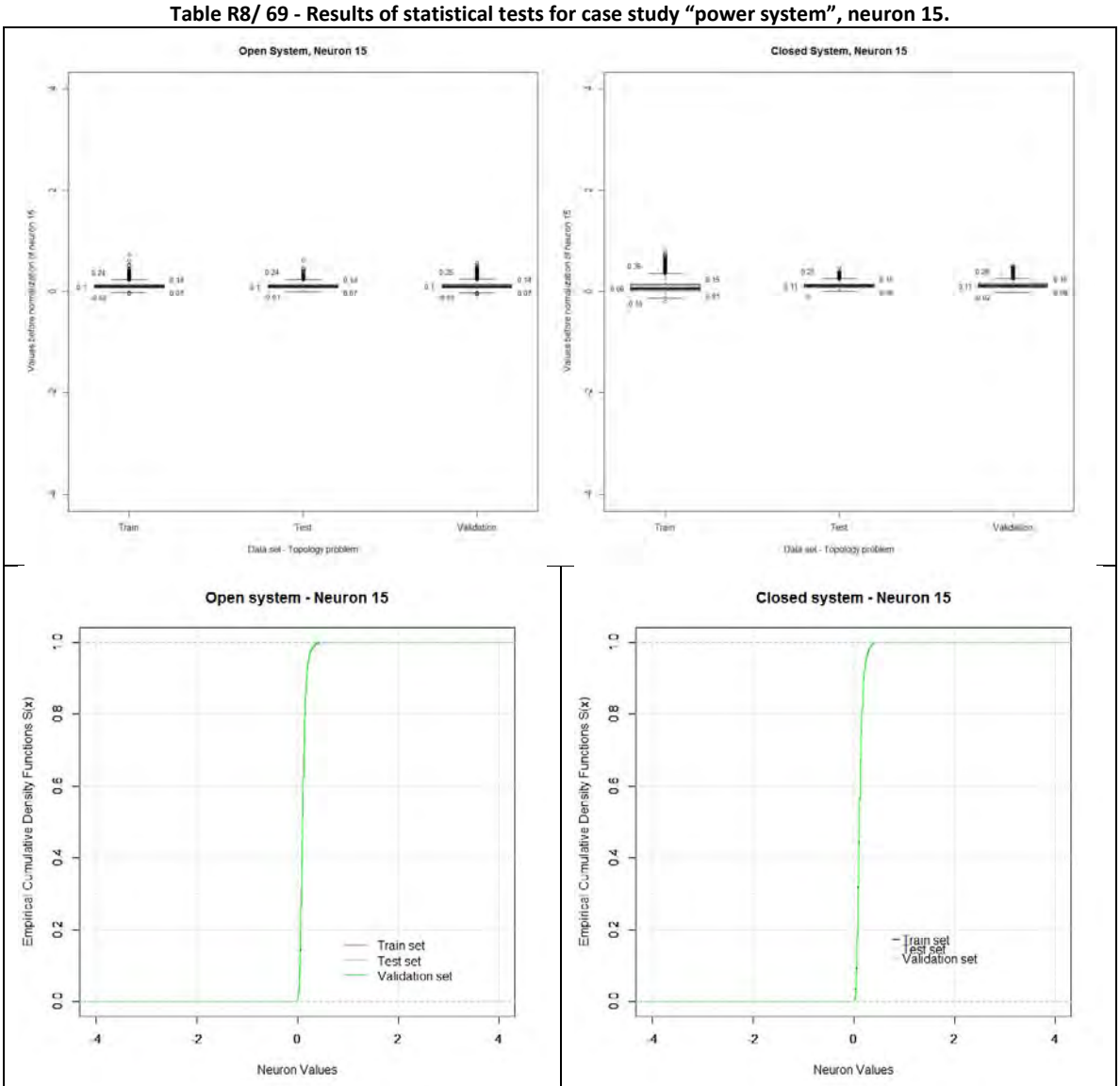
Table R8/ 68 - Results of statistical tests for case study "power system", neuron 14.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.3218	0.1811	Train - Test	0.9913	0.0314
Train-Valid	0.4227	0.1160	Train-Valid	0.9735	0.0371
Test - Valid	0.1701	0.2308	Test - Valid	0.9972	0.0192

Neuron 14 has the label Q_flow3-24 and relates to reactive power flow on the line 3-24. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 68).

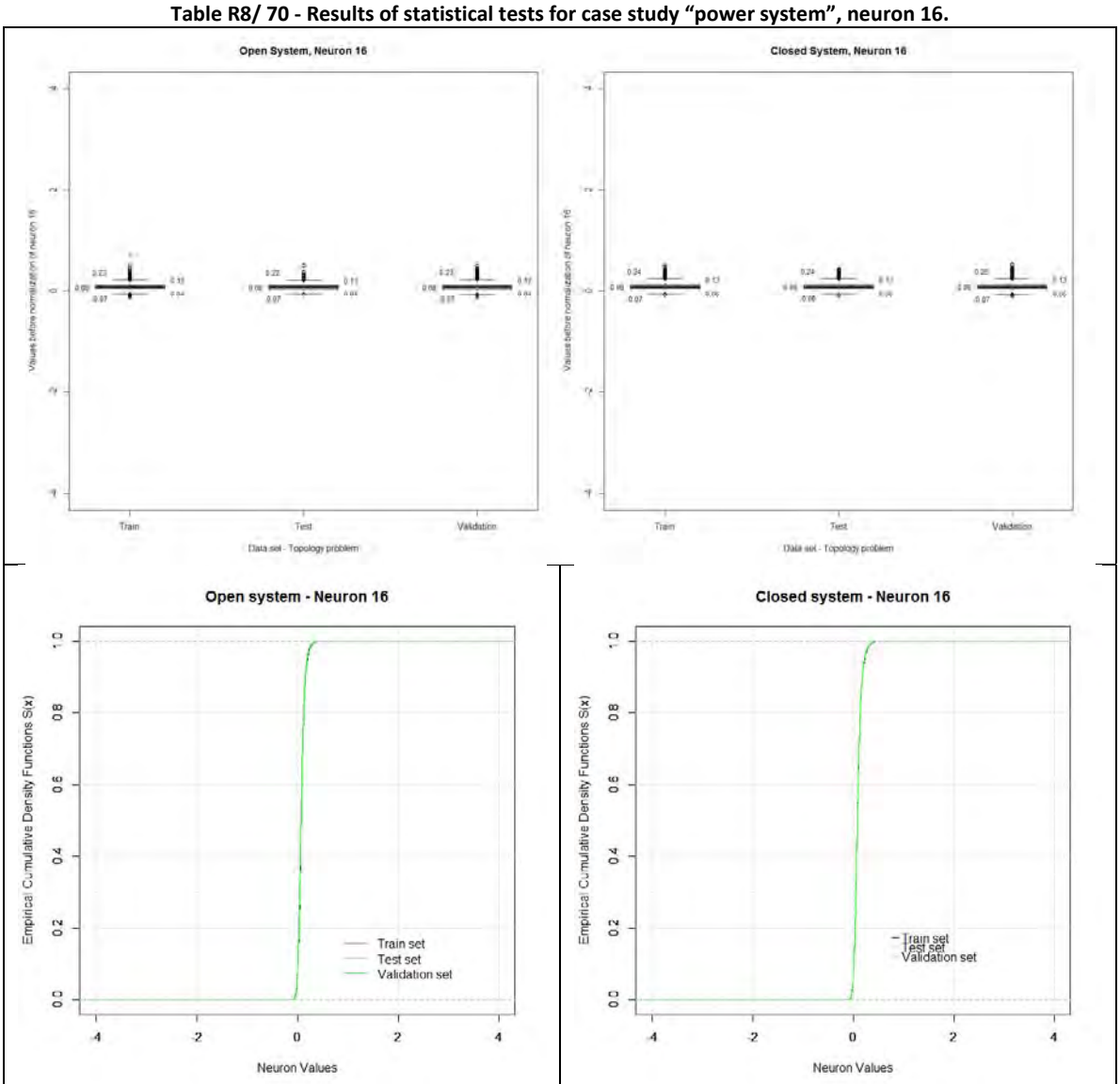
Table R8/ 69 - Results of statistical tests for case study "power system", neuron 15.



Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.5915	0.0852	Train - Test	0.9714	0.0267
Train-Valid	0.0415	0.5271	Train-Valid	0.2629	0.2229
Test - Valid	0.7913	0.1371	Test - Valid	0.5962	0.0779

Neuron 15 has the label Q_flow4-9 and relates to reactive power flow on the line 4-9. The null hypothesis was rejected for the case Train-Validation within the open system with both tests, which indicates the non-similarity of this pair of data sets, considering a 5% significance level. For all the remainder cases, the null hypothesis was accepted by both tests, indicating the population similarity among each pair of data sets, also with a 5% significance level (see Table R8/ 69).

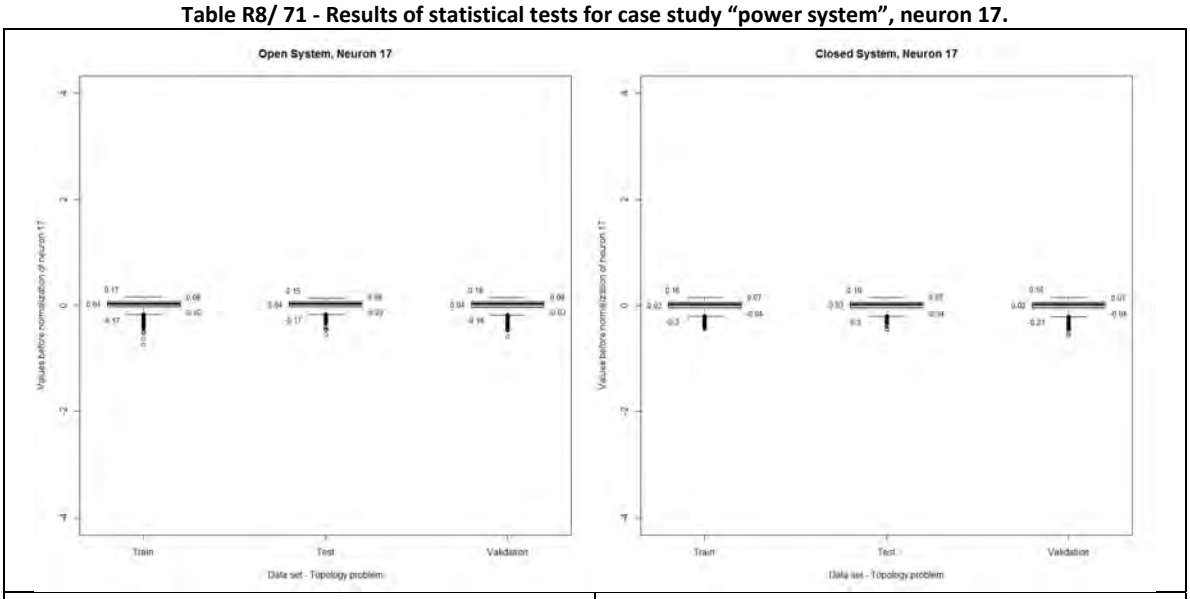
Table R8/ 70 - Results of statistical tests for case study "power system", neuron 16.

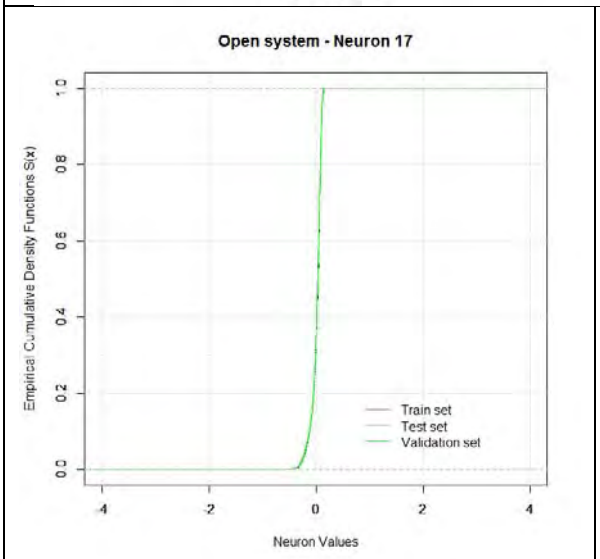


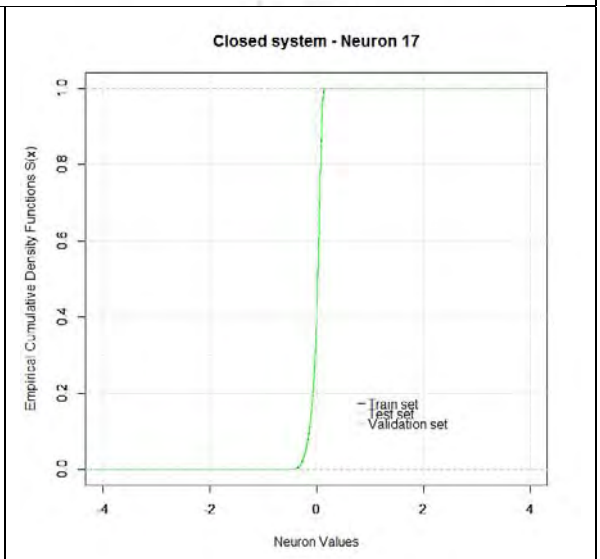
Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.4271	0.1558	Train - Test	0.6284	0.0800
Train-Valid	0.7640	0.0836	Train-Valid	0.4147	0.1186
Test - Valid	0.2595	0.1540	Test - Valid	0.3300	0.1450

Neuron 16 has the label Q_flow8-9 and relates to reactive power flow on the line 8-9. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 70).

Table R8/ 71 - Results of statistical tests for case study "power system", neuron 17.



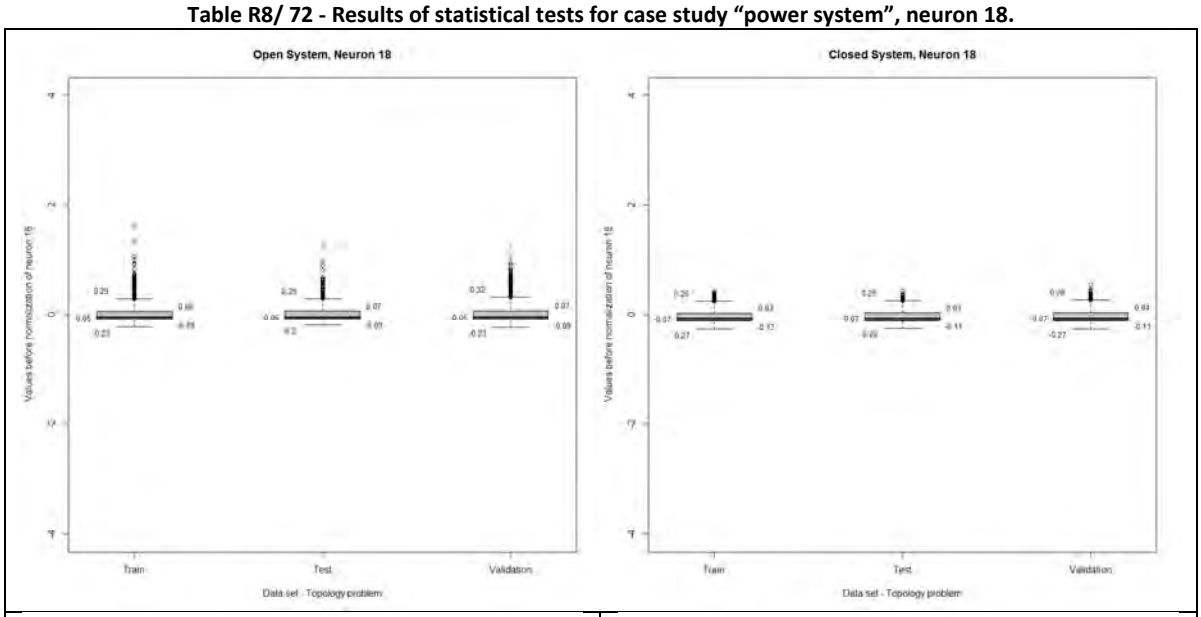


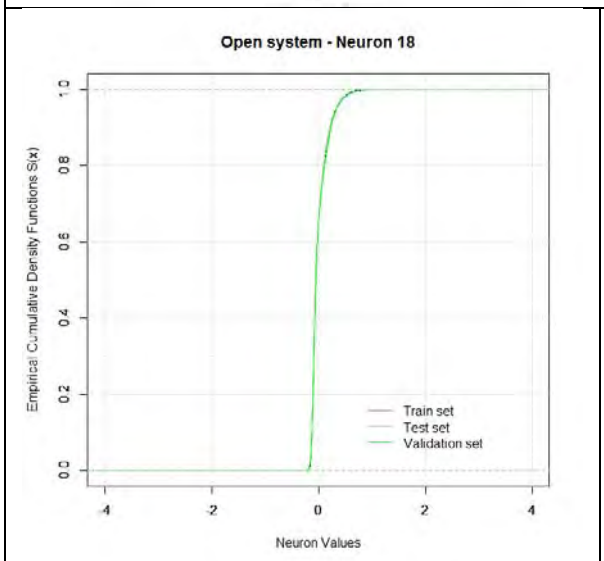


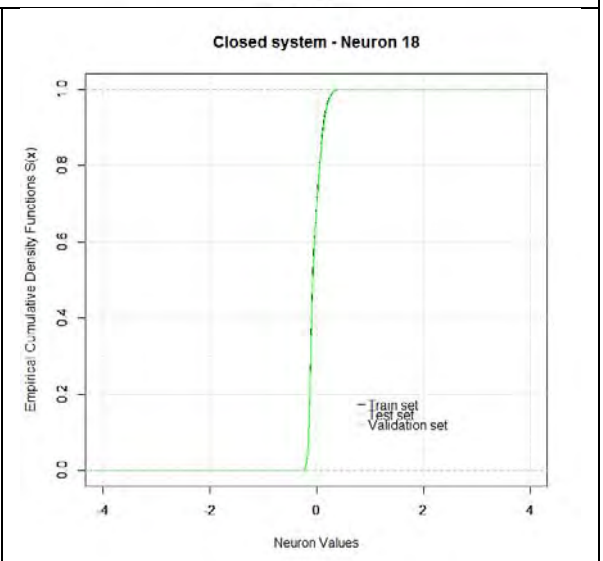
Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.8760	0.0449	Train - Test	0.9746	0.0279
Train-Valid	0.1373	0.4149	Train-Valid	0.3525	0.2203
Test - Valid	0.7364	0.1111	Test - Valid	0.5640	0.1699

Neuron 17 has the label Q_flow9-11 and relates to reactive power flow on the line 9-11. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 71).

Table R8/ 72 - Results of statistical tests for case study "power system", neuron 18.







Open	Smirnov p-value	CvM TS	Closed	Smirnov p-value	CvM TS
Train - Test	0.4264	0.1005	Train - Test	0.5105	0.1514
Train-Valid	0.3906	0.2399	Train-Valid	0.1115	0.3996
Test - Valid	0.2133	0.2037	Test - Valid	0.8237	0.0363

Neuron 18 has the label Q_flow9-12 and relates to reactive power flow on the line 9-12. The null hypothesis is accepted in all cases for both tests, considering a significance level of 5%. Therefore, the population similarity is assumed among all the data sets considered (see Table R8/ 72).

83 5 Conclusions

This report includes the statistical test performed to assess the similarity of distributions composing the datasets of train, test and validation applied to test the accuracy of the ITL neural networks developed. The statistical test found to adequately address this issues were Smirnov and Crámer von Mises. These tests

were conducted under a neuron basis. The summary on the acceptance of null Hypothesis for each case was provided in Table R8/ 3, Table R8/ 28 and Table R8/ 54.

84 Bibliography

Conover, W. (1980). *Practical nonparametric statistics* (2nd ed.). New York: John Wiley & Sons.

IEEE RTS Task Force of APM Subcommittee. (1979). IEEE Reliability Test System. *IEEE Transactions on PAS, 98*(6), 2047-2054.