Speeding up Neural Network Training via Topology Sparsification

Επιτάχυνση Εκπαίδευσης Νευρωνικών Δικτύων με Αραιοποίηση της Τοπολογίας

Koultouki Marianna

Supervisor: Assoc. Prof. Katsaros Dimitrios

2nd committee member: **Assoc. Prof.** Korakis Athanasios

3nd committee member: Tousidou Eleni

A thesis submitted in fulfillment of the requirements
for the degree of Diploma Thesis
in the
Department of Electrical and Computer Engineering
University of Thessaly

Volos, Sept 2019

Dedicated to my family

Επιτάχυνση Εκπαίδευσης Νευρωνικών Δικτύων με Αραιοποίηση της Τοπολογίας

Περίληψη

Στόχος της παρούσας διπλωματικής εργασίας είναι η ταχύτερη εκπαίδευση ενός νευρωνικού δικτύου, με την τεχνική της αραιοποίησης της τοπολογίας του. Οι προηγούμενες υλοποιήσεις εφάρμοσαν αυτή την τεχνική σε ένα τυχαίο γράφημα (Erdos-Renyi model), με σκοπό τη δημιουργία μιας πιο δομημένης τοπολογίας κατά τη διάρκεια της εκπαίδευσης του δικτύου αυτού. Στην παρούσα εργασία, μελετήθηκε η απόδοση ενός νευρωνικού δικτύου, του οποίου η τοπολογία ακολουθεί τους νόμους πιο δομημένων δικτύων εξαρχής και έπειτα, κατά τη διάρκεια της εκπαίδευσης του, ανακατασκευάζεται σε ένα εξίσου δομημένο δίκτυο, όπως το scale-free και small-world. Πρόκειται για δύο σημαντικές τεχνικές για την ανάπτυξη και τη σχεδίαση δικτύων, τόσο στο χώρο της τεχνολογίας, όσο και στον πραγματικό κόσμο. Για τη διεξαγωγή των πειραμάτων μας, χρησιμοποιήθηκε ένα νευρωνικό δίκτυο τύπου MLP (Multi-Layer Perceptron), με ένα επίπεδο εισόδου, ένα εξόδου και τρία κρυμμένα επίπεδα των χιλίων νευρώνων. Δημιουργήθηκαν 5 υλοποιήσεις (Scale-Free to SET, Scale-Free to Scale-Free, Scale-Free to Scale-Free (5 strongest nodes), Scale-Free to Small-World και Small-World to Small-World), οι οποίες διαφέρουν μεταξύ τους στην αρχική και τελική διαμόρφωση της τοπολογίας του δικτύου και χρησιμοποιούν τον αλγόριθμο back propagation για την αναδιαμόρφωση των τιμών των βαρών. Στους αλγορίθμους αυτούς δόθηκε βαρύτητα τόσο στην ακρίβεια όσο και στο χρόνο που απαιτείται, ώστε το νευρωνικό δίκτυο να εκπαιδευτεί με βάσει τα δεδομένα που του δίνονται κάθε φορά. Εκτιμήθηκε πως οι καλύτερες περιπτώσεις (τόσο σε χρόνο όσο και σε ακρίβεια) αλγορίθμων παρουσιάστηκαν στις scale-free υλοποιήσεις, κατά τις οποίες, η σύνδεση των κόμβων του δικτύου δεν είναι τυχαία αλλά ακολουθεί τέτοια κατανομή, ώστε οι δημοφιλέστεροι κόμβοι να ευνοούνται (ακρίβεια έως και 92% σε 15 λεπτά). Ωστόσο, στις περιπτώσεις όπου εφαρμόστηκε η τεχνική των small-world γραφημάτων, των οποίων η δομή είναι πιο τυχαία, είχαμε χαμηλή ακρίβεια (όχι πάνω από 70%) και τεράστιο χρόνο εκτέλεσης (έως και 12 ώρες για μεγάλα αρχεία εισόδου). Τέλος, τα πειράματά μας έγιναν για 5 διαφορετικά αρχεία δεδομένων, τα οποία περιέχουν βιολογικά δεδομένα ή δεδομένα εικόνων και είναι κωδικοποιημένα με την one-hot κωδικοποίηση.

Speeding up Neural Network Training via Topology Sparsification

Abstract

The purpose of this project is to speed up the training of a neural network by using topology sparsification technique. The previous implementations applied this technique to a random graph (Erdos-Renyi model), with the aim of creating a more structured topology during the training of this network. In the present thesis, we studied the performance of a neural network, whose topology follows the laws of more structured networks from the beginning, and it is reconstructed to a similarly-structured one, during the training phase, such as scale-free and small-world. It's about two important techniques for the development and design of networks, both in technology and in the real world. In order to conduct our experiments, a MLP (Multi-Layer Perceptron) network was used, with an input layer, an output layer and three hidden layers of thousands of neurons. So, we have created 5 variants of our concept (Scale-Free to SET, Scale-Free to Scale-Free, Scale-Free to Scale-Free (5 strongest nodes), Scale-Free to Small-World και Small-World to Small-World), which differ from each other in the initial and final configuration of the network topology and back propagation algorithm was used in order for the weight values to be adjusted. These algorithms focus on both the accuracy and the time required for the neural network to be trained, regarding the data given to them, each time. It was estimated that the best cases (both time and accuracy) of algorithms are presented in scale-free implementations, in which the connection of the nodes of the network is not random but follows such distribution so that the most popular nodes are favored (up to 92% in 15 minutes). However, when the technique of small-world graphs is applied, whose structure is more random, we have low accuracy (not over 70%) and enormous execution time (up to 12 hours regarding large datasets). Finally, our experiments were done, using 5 different datasets, which contain biological or image data, being encoded with one-hot encoding.

Acknowledgments

Firstly, I would like to express my thanks and gratitude to my supervisor Dimitrios Katsaros, for his constructive suggestions during the planning and development of this thesis. He was always available when I needed his opinion and advice.

I am also grateful to Prof. Korakis Athanasios and Tousidou Eleni for their valuable help and support.

Furthermore, I am deeply thankful to my friend Euaggelia Fragkou for our excellent collaboration to this research work.

In addition, I would like to thank all my friends for their support and all the unforgettable moments we have experienced in Volos. These past five years have been amazing!

Finally, I wish to thank my family for their continuous support throughout the years, unconditional love and understanding. They taught me how to overcome the difficulties and never give up!

Contents

Περίληψη 3

Abstract 4

Acknowledgments 6

Chapter 1 17

Introduction 17

1.1 Machine Learning and Deep Learning 17

1.2 Applications of Deep learning in last years 17

1.3 Background on Neural Networks 20

1.4 Introduction to Multi-Layer Perceptron neural network (MLP) 26

1.5 Technical background 28

1.6 Motivation and contributions 32

Chapter 2 33

Related Work 33

Chapter 3 39

Network Science 39

3.1 Techniques of Network construction 39

3.1.1 Regular Lattices 39

3.1.2 Random Network 41

3.1.3 Small World Network 43

3.1.4 Scale Free Network 47

Chapter 4 51

The Proposed Techniques 51

4.1 Sparse Evolutionary Training (SET) algorithm 51

4.2 Data formatting 52

4.3 The Proposed Techniques 53

4.3.1 Scale Free to SET 53

4.3.2 Scale Free to Scale Free 55

4.3.3 Scale Free to Scale Free (5 strongest nodes) 56

4.3.4 Scale Free to Small World 58

4.3.5 Small World to Small World 59

Chapter 5 61

Experimental evaluation 61

5.1 Evaluation settings 61
5.1.1 Dataset information 61
5.1.2 Specific variable values and software environment 63
5.2 Experimental results 63
5.2.1 SET 63
5.2.2 Scale Free to SET 71
5.2.3 Scale Free to Scale Free 80
5.2.4 Scale Free to Scale Free (5 strongest nodes) 88
5.2.5 Scale Free to Small World 97
5.2.6 Small World to Small World 105
5.3 Results - Discussion 112

Chapter 6 **115**
Conclusion 115

REFERENCES **116**

List of Figures

1.1 Step function 0/1.....21
1.2 A simple neural network model.....21
1.3 Linear (A) vs. Non-Linear (B) problems.....22
1.4 main process of CNN.....23
1.5 Restricted Boltzmann Machine.....24
1.6 A simple RNN.....25
1.7 Categorization of neural algorithms.....26
1.8 Three layer network.....27
1.9 ReLU activation function.....31
1.10 FReLU activation function.....31
1.11 Differences between ReLU (a) and FReLU (b).....31
2.1 Dropout Neural Net Model.....35
2.2 An illustration of meProp method.....35
3.1 A regular lattice.....40
3.2 A non-regular lattice.....41
3.3 Illustration of a regular lattice.....41
3.4 Classical Erdos-Renyi model.....42
3.5 Bell Curve Distribution of Node Linkages.....42
3.6 Small-world network algorithm.....44
3.7 Increasing randomness.....45
3.8 Watts and Strogatz's small-world model with p=0.01102.....46
3.9 Watts and Strogatz's small-world model with p=0.02009.....46
3.10 Watts and Strogatz's small-world model with p=0.06669.....47
3.11 Poisson Distribution vs. Power-Law Distribution for k nodes.....48
3.12 Scale-Free network algorithm 49
4.1 SET pseudocode.....52
5.1 SET accuracy, using ReLU activation function and lung.mat file.....64
5.2 SET accuracy, using FReLU activation function and lung.mat

file.....64
5.3 SET accuracy, using ReLU activation function and lung_discrete.mat file.....65
5.4 SET accuracy, using FReLU activation function and lung_discrete.mat file.....65
5.5 SET accuracy, using ReLU activation function and TOX_171.mat file.....66
5.6 SET accuracy, using FReLU activation function and TOX_171.mat file.....67
5.7 SET accuracy, using ReLU activation function and CLL_SUB_111.mat file.....68
5.8 SET accuracy, using FReLU activation function and CLL_SUB_111.mat file.....68
5.9 SET accuracy, using ReLU activation function and COIL20.mat file.....69
5.10 SET accuracy, using FReLU activation function and COIL20.mat file.....70
5.11 Scale Free Set accuracy, using ReLU activation function and lung.mat file.....71
5.12 Scale Free Set accuracy, using FReLU activation function and lung.mat file.....71
5.13 Scale Free Set accuracy, using ReLU activation function and lung_discrete.mat file.....73
5.14 Scale Free Set accuracy, using FReLU activation function and lung_discrete.mat file.....74
5.15 Scale Free Set accuracy, using ReLU activation function and TOX_171.mat file.....75
5.16 Scale Free Set accuracy, using FReLU activation function and TOX_171.mat file.....75
5.17 Scale Free Set accuracy, using ReLU activation function and CLL_SUB_111.mat file.....76
5.18 Scale Free Set accuracy, using FReLU activation function and

CLL_SUB_111.mat file.....77
5.19 Scale Free Set accuracy, using ReLU activation function and COIL20.mat file.....78
5.20 Scale Free Set accuracy, using FReLU activation function and COIL20.mat file.....78
5.21 Scale Free to Scale Free accuracy, using ReLU activation function and lung.mat file.....80
5.22 Scale Free to Scale Free accuracy, using FReLU activation function and lung.mat file.....80
5.23 Scale Free to Scale Free accuracy, using ReLU activation function and lung_discrete.mat file.....82
5.24 Scale Free to Scale Free accuracy, using FReLU activation function and lung_discrete.mat file.....82
5.25 Scale Free to Scale Free accuracy, using ReLU activation function and TOX_171.mat file.....83
5.26 Scale Free to Scale Free accuracy, using FReLU activation function and TOX_171.mat file.....84
5.27 Scale Free to Scale Free accuracy, using ReLU activation function and CLL_SUB_111.mat file.....85
5.28 Scale Free to Scale Free accuracy, using FReLU activation function and CLL_SUB_111.mat file.....85
5.29 Scale Free to Scale Free accuracy, using ReLU activation function and COIL20.mat file.....86
5.30 Scale Free to Scale Free accuracy, using FReLU activation function and COIL20.mat file.....87
5.31 Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and lung.mat file.....88
5.32 Scale Free to Scale Free (5 strongest nodes) accuracy, using FReLU activation function and lung.mat file.....88
5.33 Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and lung_discrete.mat file.....90
5.34 Scale Free to Scale Free (5 strongest nodes) accuracy,

using FReLU activation function and lung_discrete.mat file.....90
5.35 Scale Free to Scale Free (5 strongest nodes) accuracy,
using ReLU activation function and TOX_171.mat file.....91
5.36 Scale Free to Scale Free (5 strongest nodes) accuracy,
using FReLU activation function and TOX_171.mat file.....92
5.37 Scale Free to Scale Free (5 strongest nodes) accuracy,
using ReLU activation function and CLL_SUB_111.mat file.....93
5.38 Scale Free to Scale Free (5 strongest nodes) accuracy,
using FReLU activation function and CLL_SUB_111.mat file.....94
5.39 Scale Free to Scale Free (5 strongest nodes) accuracy,
using ReLU activation function and COIL20.mat file.....95
5.40 Scale Free to Scale Free (5 strongest nodes) accuracy,
using FReLU activation function and COIL20.mat file.....95
5.41 Scale Free to Small World accuracy,
using ReLU activation function, lung.mat file and p=0.02.....97
5.42 Scale Free to Small World accuracy,
using FReLU activation function, lung.mat file and p=0.02.....97
5.43 Scale Free to Small World accuracy,
using ReLU activation function, lung.mat file and p=0.075.....99
5.44 Scale Free to Small World accuracy,
using FReLU activation function, lung.mat file and p=0.075.....99
5.45 Scale Free to Small World accuracy,
using ReLU activation function, COIL20.mat file and p=0.02.....101
5.46 Scale Free to Small World accuracy,
using FReLU activation function, COIL20.mat file and p=0.02.....101
5.47 Scale Free to Small World accuracy,
using ReLU activation function, COIL20.mat file and p=0.075.....103
5.48 Scale Free to Small World accuracy,
using FReLU activation function, COIL20.mat file and p=0.075.....103
5.49 Small World to Small World accuracy,
using ReLU activation function, lung.mat file and p=0.02.....105
5.50 Small World to Small World accuracy,

using FReLU activation function, lung.mat file and p=0.02.....105
5.51 Small World to Small World accuracy,
using ReLU activation function, lung.mat file and p=0.075.....107
5.52 Small World to Small World accuracy,
using FReLU activation function, lung.mat file and p=0.075.....107
5.53 Small World to Small World accuracy,
using ReLU activation function, COIL20.mat file and p=0.02.....108
5.54 Small World to Small World accuracy,
using FReLU activation function, COIL20.mat file and p=0.02.....109
5.55 Small World to Small World accuracy,
using ReLU activation function, COIL20.mat file and p=0.075.....110
5.56 Small World to Small World accuracy,
using FReLU activation function, COIL20.mat file and p=0.075.....111
5.57 Accuracy evaluation of five algorithms, including SET code,
using five different datasets.....113
5.58 Time evaluation of five algorithms, including SET code, using
five different datasets.....114

List of Tables

5.1 Datasets characteristics.....61
5.2 Statistics of SET algorithm using lung.mat file.....64
5.3 Statistics of SET algorithm using lung_discrete.mat file.....66
5.4 Statistics of SET algorithm using TOX_171.mat file.....67
5.5 Statistics of SET algorithm using CLL_SUB_111.mat file.....68
5.6 Statistics of SET algorithm using COIL20.mat file.....70
5.7 Statistics of Scale Free to Set algorithm, using lung.mat file.....72
5.8 Statistics of Scale Free to Set algorithm, using lung_discrete.mat file.....74
5.9 Statistics of Scale Free to Set algorithm, using TOX_171.mat file.....76
5.10 Statistics of Scale Free to SET algorithm, using CLL_SUB_111.mat file.....77
5.11 Statistics of Scale Free to SET algorithm, using COIL20.mat file.....79
5.12 Statistics of Scale Free to Scale Free algorithm, using lung.mat file.....81
5.13 Statistics of Scale Free to Scale Free algorithm, using lung_discrete.mat file.....83
5.14 Statistics of Scale Free to Scale Free algorithm, using TOX_171.mat file.....84
5.15 Statistics of Scale Free to Scale Free algorithm, using CLL_SUB_111.mat file.....86
5.16 Statistics of Scale Free to Scale Free algorithm, using COIL20.mat file.....87
5.17 Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using lung.mat file.....89
5.18 Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using lung_discrete.mat file.....91

5.19 Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using TOX_171.mat file.....92
5.20 Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using CLL_SUB_111.mat file.....94
5.21 Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using COIL20.mat file.....96
5.22 Statistics of Scale Free to Small World algorithm, using lung.mat file and p=0.02.....98
5.23 Statistics of Scale Free to Small World algorithm, using lung.mat file and p=0.075.....100
5.24 Statistics of Scale Free to Small World algorithm, using COIL20.mat file and p=0.02.....102
5.25 Statistics of Scale Free to Small World algorithm, using COIL20.mat file and p=0.075.....104
5.26 Statistics of Small World to Small World algorithm, using lung.mat file and p=0.02.....106
5.27 Statistics of Small World to Small World algorithm, using lung.mat file and p=0.075.....108
5.28 Statistics of Small World to Small World algorithm, using COIL20.mat file and p=0.02.....109
5.29 Statistics of Small World to Small World algorithm, using COIL20.mat file and p=0.075.....111

Chapter 1

Introduction

1.1 Machine Learning and Deep Learning

The expectations in Artificial Intelligence have never been as high as they are today. Taking into consideration that artificial intelligence (AI) is a function that imitates the working of the human brain in processing data and in creating patterns for use in decision making, we understand that it is an emerging industry that promises revolutionary technological development [46]. Particularly speaking, machine learning is a subset of Al. It provides a method of data analysis and refers to any type of computer program that can learn by itself without having to be explicitly programmed by a human. Machine learning has been widely used in recent years in big data analytics and data mining. There are two types of learning; supervised machine learning and unsupervised machine learning. The main difference between them is that in supervised learning, the user trains the program to generate an answer based on a known and labeled data set, while in unsupervised learning the algorithms generate answers on unknown and unlabeled data. Also, supervised learning uses classification and regression algorithms, including decision trees and support vector machines (SVM), whereas unsupervised machine learning uses clustering algorithms such as K-means [36]. In this project, we emphasizing on deep learning, which is a machine learning technique process that teaches computers to learn by example. It consists of networks, capable of learning unsupervised from data, which are unstructured or unlabeled. It is quite a lot beneficial in certain types of difficult computer problems, mostly in the computer vision and natural language processing fields, by accelerating their solution. The "deep" in deep learning comes from the many layers that are built into the deep learning models, which are referred to as neural networks [36]. The success of deep learning in many areas has made neural networks among the most successful artificial intelligence methods.

1.2 Applications of Deep learning in last years

Deep learning made rapid progress in all over the last years. It was evolved in sectors like computer vision, natural language processing, automatic speech

recognition, reinforcement learning, statistical modeling, disease diagnosing, whereas it has also great impact on astrophysic or biology [28]. Machines try, through this science, to become capable of making their own decisions about how they probably react in many situations, something that is going to be a revolutionary evolution in technological fields.

It's predicted that many deep learning applications will affect our life in the near future, and particularly, within the next five to ten years, deep learning development tools, libraries, and languages will become standard components of every software development toolkit. Building cars that drive themselves (as well as full-blown self-driving cars like Google's) or constructing smart reply systems are some examples of this evolution. In order for the companies to build these types of driver-assistance services, they have to start out by training algorithms, using a large amount of data. So, by this way they can teach a computer how to take over key parts (or all) of driving using digital sensor systems instead of a human's senses. These new services could provide unexpected business models for companies and it is rumored to be on the market from 2018 and beyond. Also, AI is completely reshaping life sciences, medicine, and healthcare as an industry. Innovations in AI are enhancing the future of precision medicine and population health management in unbelievable ways [84]. One useful application is the alarm processing in emergencies. In cases of emergency, immediate evaluation and optimal corrective action are necessary. This is very difficult because the available time is not enough for the number of real-time messages (alarms) that received on the VDUs. These neural networks, used for this process, have been trained to obtain the ability of fast response [32][84]. Another popular usage areas of deep learning is voice search & voice-activated intelligent assistants (Virtual Assistants). Significant investments are already made in this area, so, voice-activated assistants can be found on nearly every smartphone. Apple's Siri is on the market since October 2011. A year after Siri, the voice-activated assistant for Android was launched by google and now the newest voice-activated intelligent assistant is Microsoft Cortana. They learn to understand your commands by evaluating natural human language to execute them. Another capability virtual assistants are endowed with, is to translate your speech to text, make notes for you, and book appointments [86]. We continue analyzing neural network applications by mentioning the ones which automatically add sounds so silent Movies. Specifically, in this task, the system must synthesize sounds to match a silent video, so it is trained using 1000 examples of video with sound of a drumstick striking different surfaces and creating different sounds. A deep learning model (which combines both convolutional neural networks and Long short-term memory (LSTM) recurrent neural networks (RNN)) associates the video frames with a database of pre-recorded sounds so as to select a sound to play that best matches what is happening in the scene. Then, the system was evaluated using a turing-test where humans recognize if the sounds in the video are real or fake. Automatic machine

translation has, also, been around for a long time; It is a task that translates words, phrase or sentence from one language, automatically into another one. By incorporating deep learning in this task we achieve better results in automatic translation of both text and images. This means that text translation can be performed without any pre-processing of the sequence, allowing the algorithm to learn the dependencies between words and their mapping to a new language. Also, automatic text generation (in which a corpus of text is learned and then new text is generated) and automatic handwriting generation (which, given a corpus of handwriting examples, generates new handwriting for a given word or phrase) are very popular deep learning applications. Also, artificial intelligence applications are made in order to process images. For example, the automatic image captioning is a task where, given an image, the system must generate a caption that describes the contents of the image [88]. A lot of deep learning algorithms were created in 2014, so that they achieve very impressive results on this problem, exploiting the potential of very good models for object classification and object detection in photographs. Advertising is also evolved by neural networks usage. For instance, deep learning helps industries make it possible for ad networks and publishers to leverage their content in order to create data-driven predictive advertising, real-time bidding (RTB) for their ads, precisely targeted display advertising and more [84]. Moreover, the News Aggregation and Fraud News Detection are created so as to help the "prospective customers" filter out all the bad and ugly news from their news feed. Deep Learning neural networks are trained and validated in order to help develop classifiers that can detect fake or biased news and remove it from your feed and warn you of possible privacy breaches. This is a very hard process, bearing in mind that the data is plagued with opinions and there is difficulty in recognizing which news are neutral or biased. Last but not least, visual recognition is a very useful application. Think for a while that you want to find an image in a huge library (for example google's library). It is very time consuming process, while using the classic searching methods. So, large-scale image Visual recognition [87] through deep neural networks is boosting growth in this segment of digital media management by using convolutional neural networks, Tensorflow, and Python extensively. Furthermore, deep learning is revolutionizing the filmmaking process as cameras learn to study human body language to imbibe in virtual characters. For instance, VEVO, Netflix, Film Making, Sports Highlights use Deep Learning. combined with face and pattern recognition, in content editing and auto-content creation, which are now a reality [86]. In conclusion, in this section we mentioned some of the extraordinary applications that has already a great impact on human lives and science evolution. So, we all understand that deep learning is changing the way we look at technologies. There is a lot of excitement around artificial intelligence, machine learning and deep learning [84]. Furthermore, it is an amazing opportunity to create a powerful innovative technology. However, rapid development in artificial intelligence, automation and

robotics raise serious questions about potential adverse human rights impacts and the future of working environment and rights of workers [85]. This is a critical and moral issue and scientists have to take always into consideration that all this technological evolution is made in order to enhance and not to worsen the quality of human lives.

1.3 Background on Neural Networks

The primary interest in research and study of neural networks came from enthusiasm for the functioning and structure of the human brain. Scientists have been excited about the way neurons operate, how neural cells coexist and effectively create a dense communication network. For this reason, they rushed to mathematical modeling. Therefore, it would be considered a great technological revolution to discover a new computational model, based on a web-like structure similar to that of the brain. This new model, later known as the *Connectionist Model*, is more suited to the creation of intelligent algorithms and of course other intelligence-related programming processes such as learning-training, memory, generalization, grouping of standards. Of course, in practice, artificial neural networks are closely related to biological neurons, as in the technological field, the very basic features of biological neurons have been simulated. However, it is a useful discovery since Artificial Neural Networks meet two conditions: a) they have parameters that can be modified, thus facilitating the learning process and b) the network is composed of many neurons, in order to achieve parallelism between the processing of data and the distribution of information. However, this model of artificial neural networks has a significant disadvantage: it is often difficult to train neural networks properly as well as to withdraw information from these ones so that they can be useful tools in development of intelligent processes [13].

First Neural Network

The first neural models appeared in the 1940s and 1950s, with an original neuron model, by the scientists McCulloch and Pitts, who described a simple model of neuron activity. The result of the neuron is either 0, indicating that the neuron is inactive or 1 suggesting that the neuron is at the maximum frequency. The neuron receives its entrances, multiplies each with its weight, and then adds the products.

$$u = \sum w_i x_i \quad , 1 < i < n$$

Then compare the result with a certain threshold represented by a real number, just like the weights of the neural and passes this result from the step function,

$$f(u) = \begin{cases} 0, & \alpha\nu \ u \leq 0 \\ 1, & \alpha\nu \ u > 0 \end{cases}$$

Figure 1.1: Step function 0/1 [13]

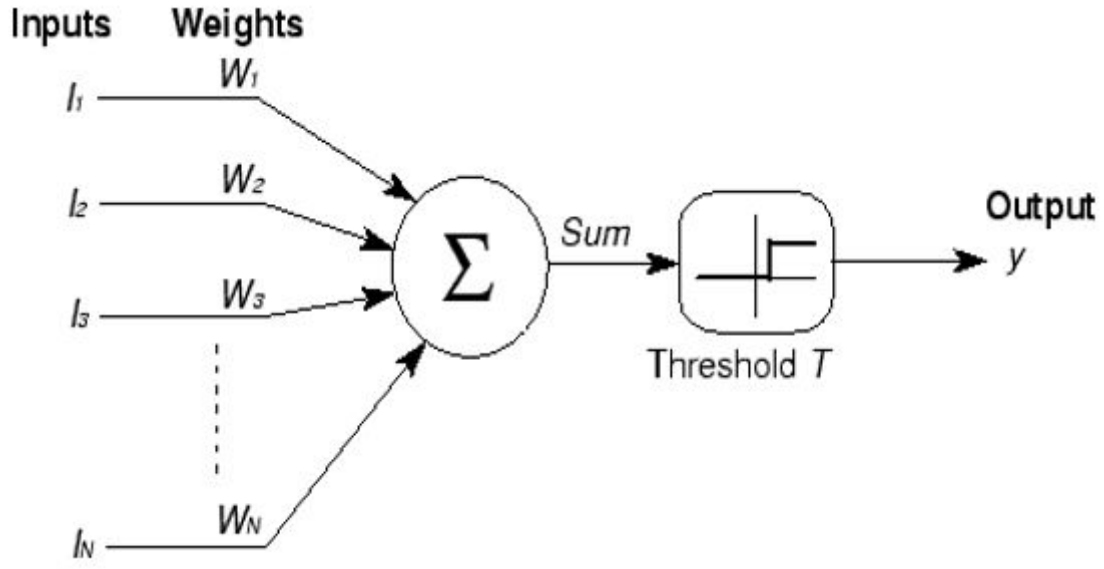


Figure 1.2: A simple neural network model [13]

Frank Rosenblatt's Perceptron Artificial Network was, also, based on the same logic. The only difference between Perceptron's and McCulloch's model was that the output of the first one would be a binary number, either in his classical form, ie (1/0), or in the bipolar form, ie (1 / -1) [13].

Moreover, through this model, Rosenblatt introduced the first *training rule* for a neural network, which has, also, been known as the fixed increment rule. What is required in this rule is to find a way to learn the system's parameters so that the goal of the neural network is achieved: a good prediction. The appropriate values of the parameters that contribute to the correct network prediction are not known, but we know the correct output of the network for each input, so that we can check if the network prediction is correct. Therefore, the network is trained with supervision, and taking into account all the information given, updates the weight values, repetitively. Specifically, the input vector is repetitively displayed on the network. A full appearance defines an epoch. This rule modifies the weights for each component of the input vector, only if the output of the network is not the

expected one (sort error).

$$y(x) = f(W(k-1)^T)X^p)$$

where f is the activation function, W is the synaptic weights, k is the current epoch, X is the input vector and p is each component, y is the output in k epoch. When the prediction of neural network isn't compatible with the right result ,already, given, this algorithm modifies the weights by adding or subtracting a percentage of the given input,

$$W(k) = W(k-1) + a(d^p - y)X^p$$

where, W(k) the corrected weights after the k iteration and a the learning rate, which determines in what way weights will be corrected and it is a small positive number. However, in 1969 Minsky and Papert proved that this model had limited potential. It has been shown that the Perceptrons (one level) artificial networks so far are only capable of learning linearly separable data and therefore could not identify many categories of data. This has been a serious disadvantage for this network, since most problems of the outside world are non-linearly separable [13].

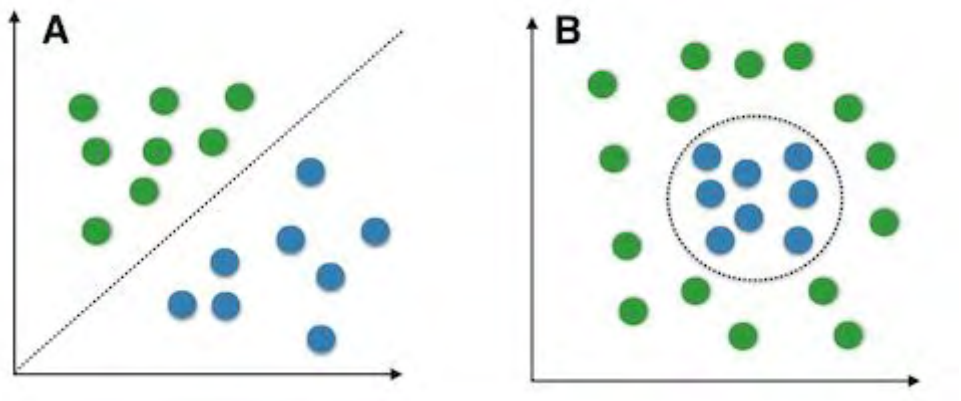


Figure 1.3: Linear (A) vs. Non-Linear (B) problems [82]

A simple example is shown in the Figure 1.3. It is obvious that no straight lines can divide the two different types of shapes, in (b), into two classes, so perceptron is not suitable to solve this problem. Research on the neural network sector has been halted for several years. In 1980, two major network models were launched and they are going to be useful in this industry: the Hopfield model and the Multi-layer-perceptron (MLP) model [13].

As passing through the years, many new kinds of neural networks made also their

appearance, like CNN (Convolutional neural network), RBM (Restricted Boltzmann machine), RNN (Recurrent neural network), etc.

CNN is known as a feed-forward neural network and it was proposed by Hubel and Wiesel in 1960. It is about an efficient recognition algorithm which is used in pattern recognition and image processing which became a hot topic in voice analysis and image recognition. CNN neural network includes two layers, one is feature extraction layer and the other is feature map layer. CNN is a multilayer network that has the special design for identification of two-dimensional image information but it has more layers (input, convolution, sample and output layer). This network implements the convolution and sampling processes. In the first process a trainable filter, deconvolution of the input image, is used in which a bias is added. After that, in sampling stage, n pixels of each neighborhood through pooling steps, become a pixel, and then by scalar weighting Wx + 1 weighted, bias bx + 1 is added, and then it passes through a transfer function producing a narrow n times feature map Sx + 1 [39].

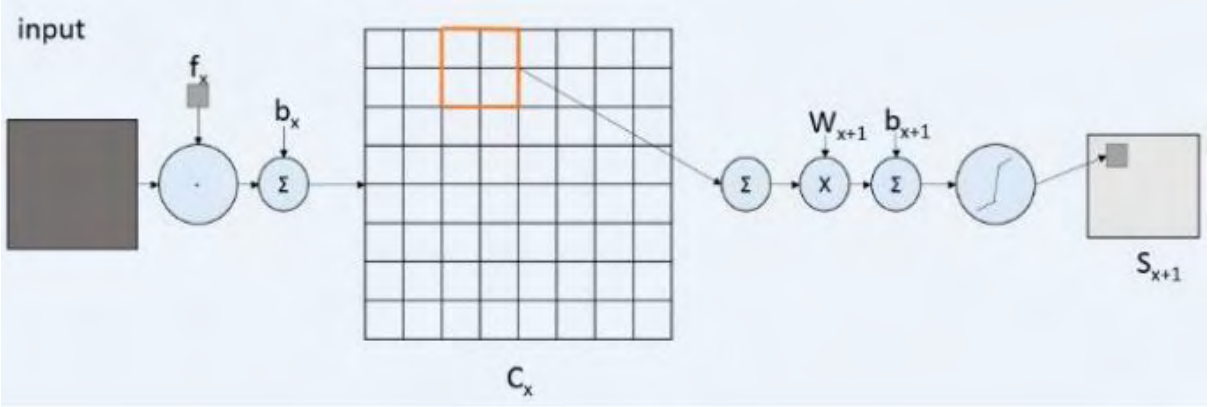


Figure 1.4: main process of CNN [39]

Instead of CNN, RBM is a neural network which has two layers with links only between these two layers of neurons. These connections going both ways (forward and backward) that have a probabilistic / energy interpretation. The lower layer is called visible and the higher is called hidden. RBM is considered an energy-based model. The global energy function of an RBM network, is given by the following equation:

$$Energy(u,h) = -b'u - c'h - h'Wu$$

where u are the values of visible neurons, h are the values of hidden neurons, b and c are biases vectors and W is the matrix of weights connections. The neurons in RBM are binary and stochastic, meaning that each neuron outputs values 0 or 1 with certain probability whose type is the following:

$$P(h_j = 1|u) = \frac{1}{1+exp(-c_j - \sum_i u_i w_{ij})} = logsig(c_j + \sum_i u_i w_{ij})$$

The Contrastive Divergence (CD) algorithm, which proposed by Hinton in 2002 is used for RBM' s training. This algorithm has the positive and the negative phase. First of all, the values of the hidden units are sampled in the first positive phase: $h_1 \sim P(h|u_1)$. From these values h_1, the reconstruction of values of the visible units is sampled: $u_2 \sim P(u|h_1)$, and so on for h_2, etc. Next step is the weights update which is implemented using the following type:

$$w_{ij}(new) = w_{ij}(old) + a(u_{1i}h_{1j} - u_{2i}P(h_{2j} = 1|u_2))$$

where a is learing rate [40].

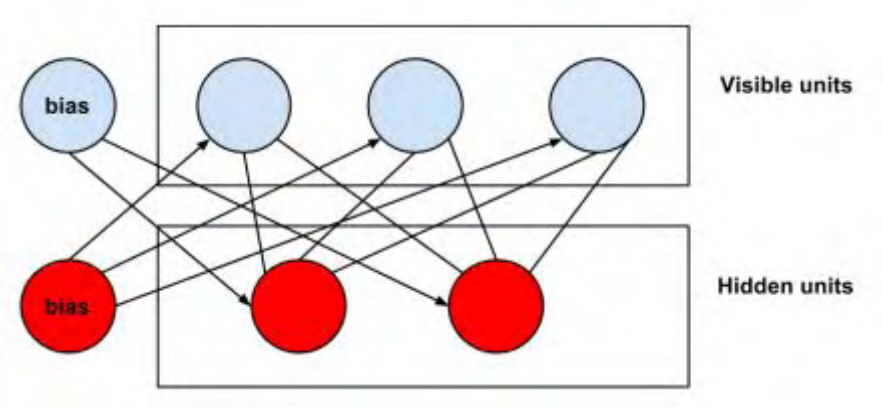


Figure 1.5: Restricted Boltzmann Machine with three visible units and two hidden units (and biases) [41]

However, CNN and RBM cannot used in prediction problems that involve sequential data. For this kind of problems Recurrent neural networks are created. RNN is the first algorithm that remembers its input due to internal memory which is very important in machine learning science. It is about a type of neural network witch has recurrent connection and is capable of modelling sequential data for sequence recognition and prediction. It has three layers which are input, recurrent (hidden/state) and output layers [43].

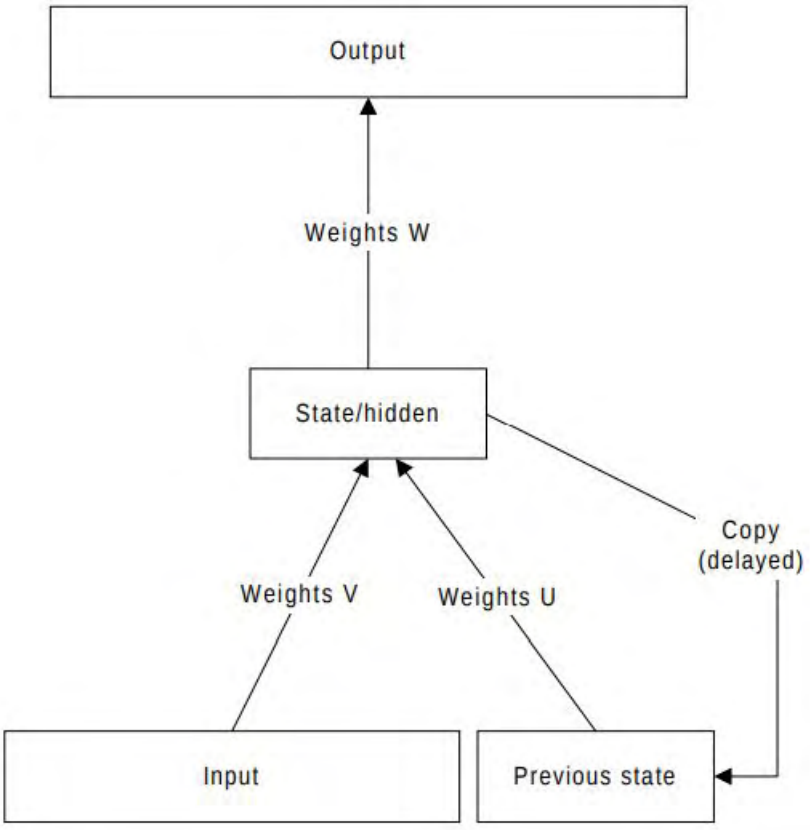


Figure 1.6: A simple RNN [44]

A simple recurrent network has activation feedback which embodies short-term memory. A hidden layer is updated not only with the external input of the network but also with activation from the previous forward propagation as shown in Figure 1.6. The feedback is modified by a set of weights as to enable automatic adaptation through learning. Backpropagation algorithm is usually used in RNNs training [44].

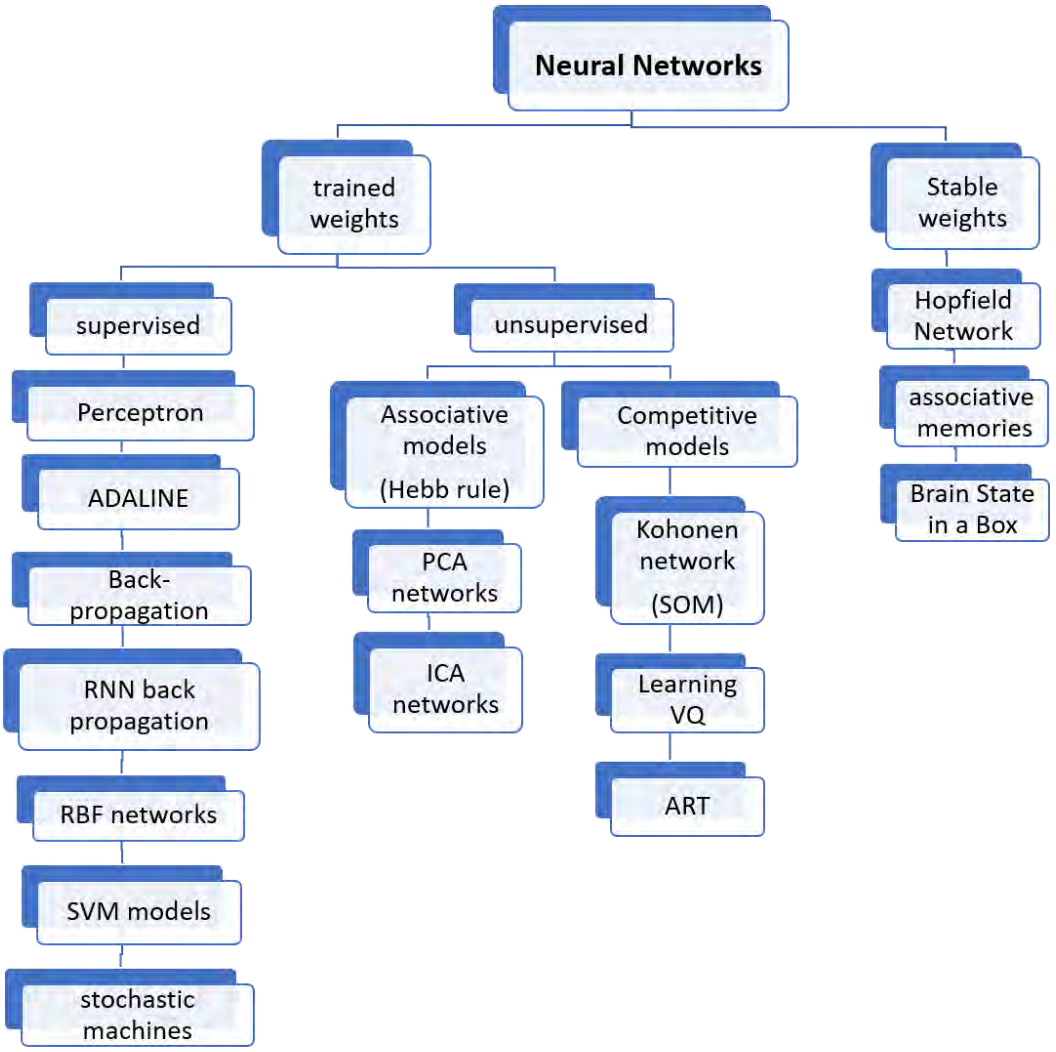


Figure 1.7: Categorization of neural algorithms [13]

So, in this present work, we will be working on a MLP (Multilayer Perceptron) feed-forward network, which has a lot of processing power and it is a suitable type of neural network, for solving classification prediction problems.

1.4 Introduction to Multi-Layer Perceptron neural network (MLP)

Different neural network structures can be constructed by using different processing elements regarding the reason what they are created for. A variety of neural network structures have been developed for signal processing, pattern recognition, control, and so on. In this project a multi-layer perceptron neural network (MLP) is used, in order to work on biological or image data and make predictions about future health problems or image recognition respectively, by learning and comparing these data. So, in this section, we describe the structure of

a MLP neural network, which is a basic model, used in a variety of modeling and optimization problems [30]. As referenced in section 1.1, neural networks with one layer perceptron cannot solve problems in which classes are not linearly separable. The use of additional layers makes the perceptron able to solve nonlinear classification problems [31]. Hence MLP structured networks are in use of. Actually, the multilayer perceptron is a feed-forward layered network of artificial neurons, where the data circulates in one way, from the input layer to the output layer. It is composed of three layers, which are the input layer, the output layer and the hidden layer. Several algorithms are used for the learning step of MLP. The common supervised learning technique is called back propagation. It is an efficient technique that is combined with stochastic gradient descent (SGD) optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function. SGD is one of many optimization methods, namely first order optimizer, meaning, that it is based on analysis of the gradient of the objective. Back propagation algorithm consists of four stages: initializing weights, feed forward, back propagation of errors and weight update [31]. It is necessary to initialize the weights before training starts. The weights are initialized either to random or zero values. In our project, we initialize weights by using random values. In feed forward stage, the output of neural network is calculated. The nodes in input and output layers have linear activation functions, while nodes in hidden layers have nonlinear transfer function. Neurons in input layer represent the input in neural network and they don't receive any information because there is no previous layer. The input value in each neuron of hidden or output layers is calculated by a specific procedure. Specifically, each neuron receives as input the sum of the products of the weights and outputs of neurons from the previous layer. Then the threshold (which is the tolerance value to error) is added in the sum and the result is passed through the activation function. Figure 1.8 shows this procedure for a three layer neural network [29].

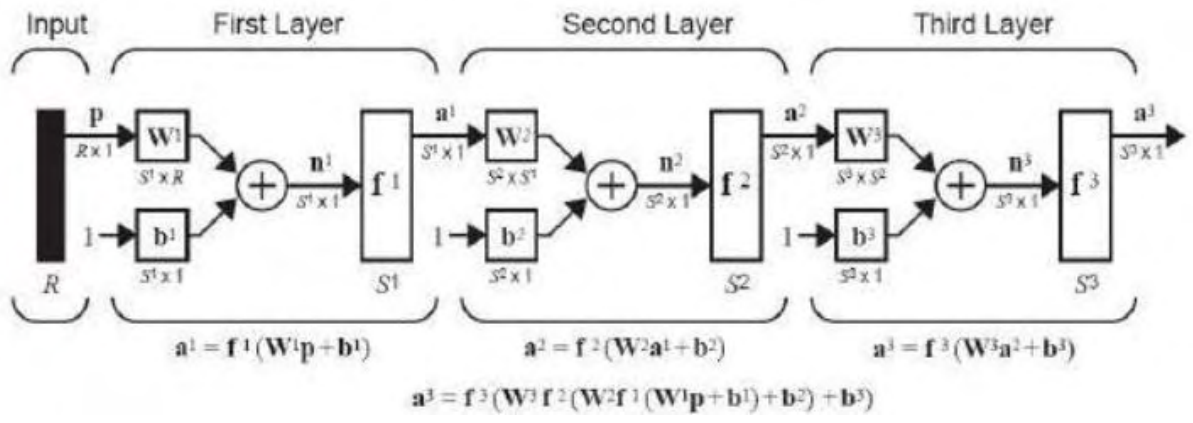


Figure 1.8: Three layer network [29]

The last stage of training, is the most time-consuming procedure of all stages in

back propagation algorithm. First of all, we have to calculate the error between the predicted output which has calculated in feed forward stage, and the correct output of neural network. The type which calculates the error in the last layer L is different from the
type for the remaining layers l = 1,...,L-1. The error in each layer of each neuron i is calculated by the following mathematical type:

- Layer L (last layer):

$$\delta_i^{(k)}(L) = f'(u_i^{(k)}(L))(d_i^{(k)} - y_i^{(k)})$$

where k is the current epoch, d_i is the correct output value of i neuron, y_i is the predicted output value of i neuron, u_i is the output of neuron i before passed through the transfer function and f' is the derivative of the activation function.

- Layers l = 1,...,L-1:

$$\delta_i^{(k)}(l) = f'(u_i^{(k)}(l)) \sum_{\mu=1}^{N(l+1)} w_{\mu i}(l+1)\delta_\mu(l+1)$$

where k is the current epoch, N is the number of neurons, $w_{\mu i}(l+1)$ is the weight ofthe link between i neuron in l layer and μ neuron in $l+1$ layer, $\delta_\mu(l+1)$ is the error of $l+1$ layer of μ neuron, u_i is the output of neuron i before passed through the transfer function and f' is the derivative of the activation function.
After the calculation of errors, the weights must be updated. For this purpose we use the following equation to find the new value of the weight between nodes i and j in each layer l :

$$w_{ij}(l,k+1) = w_{ij}(l,k) + \beta\delta_i^{(k)}(l)a_j^{(k)}(l-1)$$
$$j = 0,1,...,N(l),\ l = 1,...,L$$

where k is the current epoch, $w_{ij}(l,k)$ is the weight of link between nodes i and j in layer l, β is the learning rate, $\delta_i^{(k)}(l)$ is the error in layer l of i neuron and $a_j^{(k)}(l-1)$ is the output of j neuron in layer $l-1$ [13].

1.5 Technical background

In order to make our project statistics, we use the MSE (Mean Square Error)

method to find the accuracy of each epoch. Particularly, we find how many values our neural network predicts correctly. Then, by using the MSE, we calculate the mean Euclidean distance between the estimator and the true value [24]. Mathematically, it is given by the following type:

$$\text{MSE} = \left(\frac{1}{n}\right) \cdot \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

where n is the number of data, Y_i is the target (correct) value and $\hat{Y}_i$ is the predicted value. So, using MSE we calculate the error between the target value and the value that the neural network predicts [24].
Furthermore, there are many techniques that contributes to network training. Neural networks are trained using either batch or an on-line method. Batch normalization is a method, which is used in back propagation algorithm and it is useful and quite effective in the training of neural networks. In batch training, the weight update is calculated regarding some inputs and then it is applied to the weights, after specific number of iterations [20]. Also, momentum is a term which is used in several methods of neural network training. It is still an important factor, due to its good influence in the weight update, in cases where the gradient value is small [22]. It has the ability of improving the speed of convergence for most eigen components in the system by bringing them closer to critical damping [23].

A central problem in machine learning is supervised learning that is, learning from labeled training data. For example, a learning system for medical diagnosis might be trained with examples of patients, whose case records (medical tests, clinical observations) and diagnoses were known. Learning algorithms essentially operate by searching some space of functions for a function that fits the given data [17]. In order to minimize error on the training data (prevent overfitting), we use regularization techniques, which is obviously an important factor that controls network accuracy. There are many methods, used to regularize data. In this project, the training set size is large relative to the dimension of the input, so, some special mechanism has to be used so as to encourage the fitted parameters to be small and prevent overfitting (which is a phenomenon, typically characterized by high variance and low bias estimators referred to network performance [83]). Therefore, we end up using two standard regularization methods L1 and L2 (special for this amount of data), whose basic difference is the penalty term, added in the loss function form during the procedure of updating the parameters [21].

The L1 regularization uses a penalty term which encourages the sum of the absolute values of the parameters to be small. Especially, L1 shrinks the less

important feature coefficient to zero thus, removing some feature altogether. Consequently, this makes it a very useful method in feature selection settings, where it is known that many features should be ignored. For example, linear least squares regression with L1 regularization is called the Lasso algorithm (Tibshirani, 1996), which is known to generally give sparse feature vectors [21].

On the other hand, L2 is quite different, while it adds a square sum of a coefficient as penalty term to the loss function. The L2 regularizer, being an upward-facing convex function, can unflatten the flat regions and curve up some stationary points without severely changing the minimum locations. Briefly, L2 encourages the sum of the squares of the parameters to be small [21].
The last but most significant parameter is the activation function, selected to calculate the output of each neuron. The weighted sum of input and biases, computed by these functions, produce the total result. Activation functions, referred to as transfer functions, too, vary according to the network. Below, we define transfer functions which are used in our algorithms [27].

Here, we analyze ReLU and FReLU functions, which are used in output calculation in hidden layers and sigmoid function that is used to calculate the last output of the network.

- This activation function is often referred to as the logistic function or squashing function. It is a non-linear function which is used mostly in feedforward neural networks. Moreover, it is a bounded differentiable real function, defined for real input values, with positive derivatives everywhere and some degree of smoothness. The sigmoid function used in the output layers and it is suitable in neural networks which solves binary classification problems [27]. It is given by this mathematical type:

$$f(x) = \frac{1}{1+(exp)^{-x}}$$

- The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton and it is the most widely used activation function for training neural networks. It is considered as a fast and, most successfully, used transfer function. Comparing ReLU and sigmoid we conclude that the first one offers better performance and generalization in deep learning [27]. We use this one to produce the output in every layer. ReLU is a linear function and it is given by:

$$f(x) = max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$$

Figure 1.9: ReLU activation function [27]

- We see that ReLU provides sparsity by simply restraining the negative value to hard-zero. However, it results negative missing. Hence Suo Qiu, Xiangmin Xu and Bolun Cai proposed the Flexible rectified linear unit (FReLU) activation function in their paper [4]. FReLU adjusts the ReLU by a rectified point to capture negative information and provide zero-like property. Because of that, it offers fast convergence and higher performance, low computation cost without exponential operation, compatibility with batch normalization, etc [4], things that make it more efficient that ReLU, in some cases.

$$frelu(x) = \begin{cases} x + b_l & \text{if } x > 0 \\ b_l & \text{if } x \leq 0 \end{cases},$$

Figure 1.10: FReLU activation function [4]

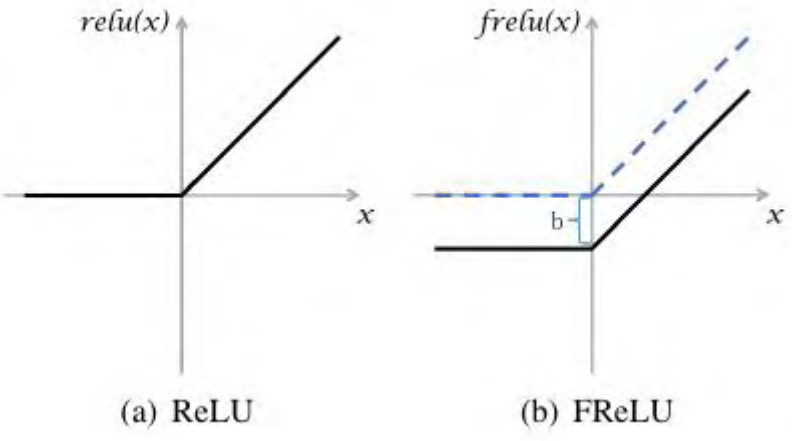


Figure 1.11: Differences between ReLU (a) and FReLU (b) [4]

Topology sparsification is a very promising technique for speeding up neural networks training. This method supports that not all the connections between the nodes are effective for the network. Some links weights have values close to zero which means that they are not give any information in the network. So these links could be removed, without influence neural network accuracy. There are some methods that use sparsification decisions after the training part, in order to speed up the training phase.
In this project we work on topology sparsification of neural networks, using tools from network science. The only prior work that investigated such an approach, is reported in [1]. They start from a randomly constructed network according to the Erdos-Renyi model and through the SET algorithm, described in chapter 4, they create a similar to scale-free structured network. Our motivation results by the fact that real world networks such as genetic networks or the World Wide Web are complex and heterogeneous networks [51] and need more complex techniques in order to be described well. More specifically we aim at producing a structure topology, based on scale-free or small-world techniques, starting from another or same structure topology (scale-free or small-world topology). This might end up being not efficient at all, because the initial network might be too dense (and almost fully connected) or too sparse. In that context, in this project we propose five algorithms which use and combine scale-free and small-world methods. All algorithms construct structured neural topologies starting from other structured neural topologies, all being different from fully connected bipartite ones in order to speed up the training time. Also, we evaluate the performance of all implemented algorithms and confirm their rationale.

The rest of the work is structured as follows: section 2 presents the related work, and section 3 briefly gives some necessary concepts from network science. Section 4 describes and proposes neural topology evolution algorithms, and in section 5 we evaluate the neural network' s classification accuracy and training time. Finally, section 6 concludes this work.

The literature on speeding up neural network training has a long history and it dates back to the late ‘80 – early ‘90. We will present the related work categorized into families of techniques; our listing is by no means extensive, but we strive to give the most representative and/or more recent members of each family.

One of the first families of acceleration methods includes members that meant to replace the traditional gradient (steepest) descent optimization method. Steepest descent is based on a first order Taylor series approximation of the performance function (mean square error) and it is very slow. Therefore, methods based on second order Taylor series were investigated, such as Newton’s method and particular adaptations of it, e.g., the Levenberg-Marquardt algorithm [62] which is much faster. Other algorithms that departed from the first order gradient concept, are those based on conjugate gradient [63], and the similar in spirit quasi-Newton method of Broyden-Fletcher-Goldfarb-Shanno (BFGS), along with its variations, e.g., L-BFGS [64]. Recently, fast optimizers have been proposed such as Adam, Adadelta, Nadam [65].

Adadelta method dynamically adapts over time using only first order information. This method has a lot of benefits such as minimal computational over gradient descent, no manual setting of a learning rate, separate dynamic learning rate per-dimension [58]. Few years later, Adam optimizer was invented by Kingma and Ba. It is just Adadelta optimization plus momentum. Adam is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. Adam algorithm is straightforward to implement and is used widely in deep learning because of many advantages. It is computationally efficient and it has little memory requirements. Also, that method can achieves good results fast. This optimizer converges much faster for multi-layer neural networks or convolutional neural networks, than any other optimizer [56, 57]. Nesterov-accelerated Adaptive Moment Estimation (Nadam) incorporates Nesterov momentum, which is more effective than vanilla momentum which is used in Adam algorithm.

Another family for accelerating neural training is that based on adopting variable learning rates. For instance, the Delta-Bar-Delta (DBD) method [66] assigns to each network parameter its own learning rate that varies at each iteration. The DBD algorithm is a heuristic approach to improve the convergence speed of the weights in artificial neural networks (ANNs) [68]. The Delta-Bar-Delta paradigm uses a learning method where each weight has its own self-adapting coefficient. It

also does not use the momentum factor of the back propagation networks. The remaining operations of the network, such as feedforward recall, are same to the normal back-propagation networks. Delta-Bar-Delta is a heuristic approach in training neural networks, because the past error values can be used to infer future calculated error values. This learning algorithm implements four heuristics regarding gradient descent; Every weight should have its own individual learning rate and every individual learning rate should adjust over time. Moreover, if the error derivative has the same sign for several consecutive steps, then increase the learning rate, whereas when the sign changes alternatively over a number of steps, then decrease the learning rate. Finally, the weights are updated, using the same formula as in Backpropagation method, except that, in this case, momentum is not used, and each weight has its own time-dependent learning rate [69]. Also, extended Delta-Bar-Delta (EDBD) and directed random search (DRS) belong to this kind of learning algorithms.

Similar in spirit is the SuperSAB method, which is an adaptive acceleration strategy for error back propagation learning. The main difference between them is that SuperSAB increases the learning rate exponentially instead of linearly, as in Delta-Bar-Delta method. This is done to take the wide range of temporarily suitable learning rates into account [71]. It can converge orders of magnitude faster than the original back propagation algorithm and it is only slightly unstable. In addition, the algorithm is very insensitive to the choice of parameter values, and has excellent scaling properties [67].

The recently introduced technique of dropout [72] constitutes the founding member of a new family, which accelerates training by randomly dropping units during training. Several adaptations of it have been proposed for various applications and various neural architectures, e.g. [73].
Dropout is a technique that addresses both these issues which are preventing overfitting and providing a way of approximately combining exponentially many different neural network architectures efficiently. The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 2.1 Units are dropped randomly. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5 [19]. To sum up, dropout can be considered as a method based on neural topology sparsification (as the one related mostly to our present work), in the sense that removing a neuron is equivalent to removing all its connections.

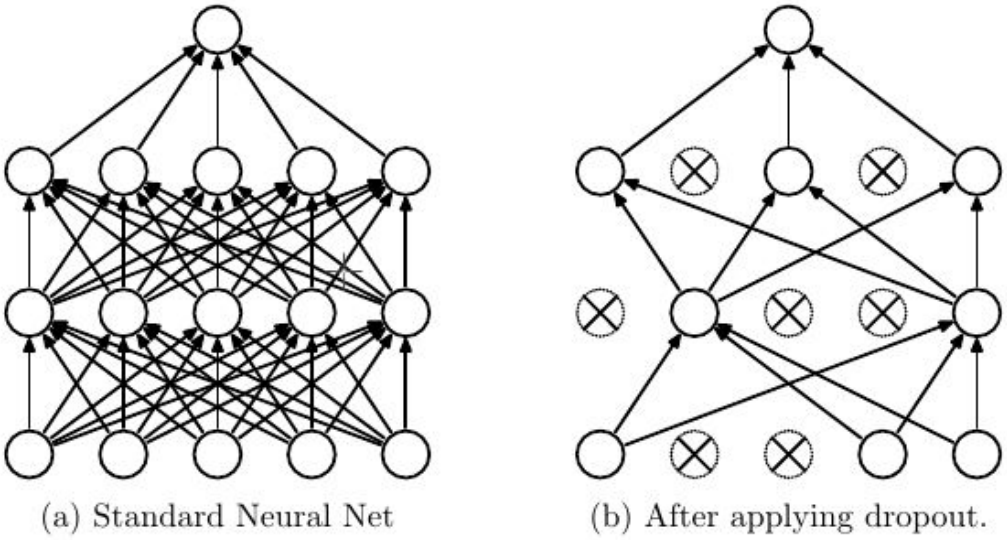


Figure 2.1: Dropout Neural Net Model. (a) A standard neural net with 2 hidden layers and (b) An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [19]

Similar in spirit, are the methods that compute only a subset of gradients during back propagation [74][75]. For example, meProp is a simple yet effective technique for neural network learning. The forward propagation is computed as usual. During back propagation, only a small subset of the full gradient is computed to update the model parameters. Subsequently, the original back propagation uses the full gradient of the output vectors to compute the gradient of the parameters, while meProp uses only top-k values of the gradient of output vector and back propagates the loss through the corresponding subset of the total model parameters [76]. Figure 2.2 shows the method meProp for a single computation unit of neural models.

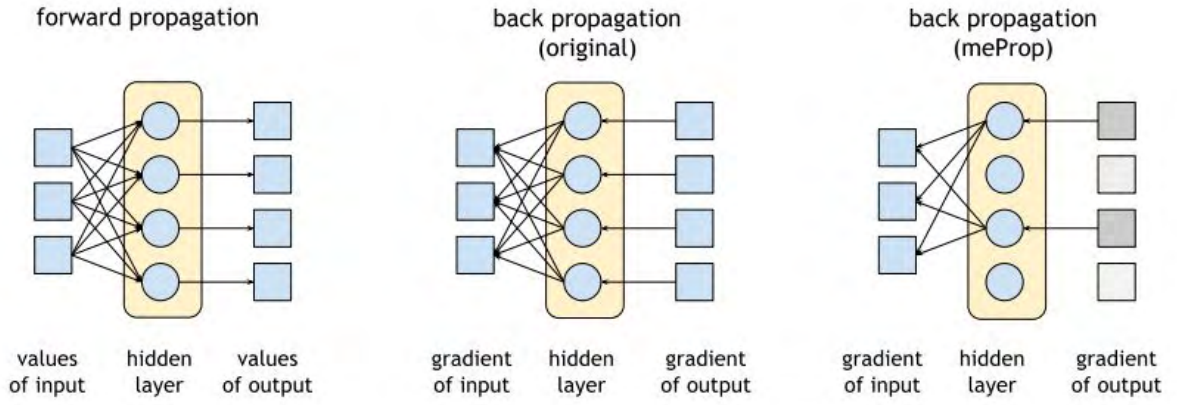


Figure 2.2: An illustration of meProp method [76]

On background, despite the popularity and success of neural networks in research, the number of resulting commercial or industrial applications have been limited. A

primary cause of this lack of adoption is due to the fact that neural networks are usually implemented as software running on general-purpose processors. As a result, training large networks for real-world applications, often takes a lot of time (weeks) [47]. It should be noted that neural networks are composed of an interconnected network of independent processing elements, and therefore, are intrinsically parallel. Hence, one of the first families of acceleration methods includes hardware implementations. Architectures such as FPGAs [77], multicore CPUs [78], TPUs [79] are increasingly used for neural training and inference.

Field Programmable Gate Arrays (FPGAs) play an increasingly important role in data sampling and processing industries because of its highly parallel architecture, low power consumption, and flexibility in classic algorithms. Especially, in the artificial intelligence field, high energy efficiency hardware and massively parallel computing capacity are demanded, concerning networks training and implementation [80].

Furthermore, a GPU implementation can achieve superior performance by taking advantage of this parallelism. Depending on the network topology (which is the arrangement of the elements, such as links and nodes of a communication network [8]), training and classification on the GPU performs faster than on the CPU. Furthermore, the GPU version scales much better than the CPU implementation with respect to the network size [48]. For instance, due to the parallel nature of neural networks, CUDA programming is a very attractive method for performance gain [47]. However, CUDA combined with different kinds of networks produces different results. For example, Researchers from Soongsil University in Korea [47] tried to implement a combination of CUDA and OpenMP in their attempt to speed up their feedforward neural network. They claimed that CUDA can indicate better performance while neural network is used for image processing. In cases of sophisticated processing problems, CUDA may not be ideal for [47]. Thus, probably, the biggest drawback of CUDA is its limitation to the NVIDIA hardware, but future languages like OpenCL [49] or DirectX 11 Compute Shader [50] will solve this problem. Until then, this technique for network accelerating, is not suitable for all network types, bearing in mind, for example, that there aren't any similar efforts for CNNs implementation, in contrast to other classifiers like Support Vector Machines (SVMs) [48].

Moreover, starting as early as 2006, Google considered deploying GPUs, FPGAs, or custom application-specific integrated circuits (ASICs) in its data centers [79][81]. The existence of few applications that could run on special hardware, could be done virtually for free using the excess capacity of Google large data centers, something that was difficult to improve on free [81]. Hence, Google creates Tensor Processor Unit (TPU), which is designed as a matrix processor specialized for neural network work loads in order to improve cost-performance by 10X over GPUs. Given this mandate, the TPU was designed, verified, built, and deployed in datacenters in just 15 months [81]. Software, running on TPUs is compatible to

GPUs and CPUs. Therefore,it is an interesting innovation that plays an important role in deep learning theory evolution.

Overall, the methods developed, in this work can be used in conjunction with any member of any family described above to accelerate training. This fact establishes a research avenue for the future. We take here a first step in walking this avenue.

Before we start with the actual implementations of our algorithms which model networks in Python, based on graphs theory, we want to devote ourselves to the origins of this theory.

Background on Network Science Concepts, Network science is the discipline that analyzes the properties and function of complex networks, such as technological, social, biological, and physical and so on. Complex network analysis consists of algorithms and methodologies for studying and developing: centralities [52], communities [53], diffusion processes [54], network growth and the respective models [55], etc. Well-studied network models comprise random networks, regular lattices, small-world networks, and scale-free networks.

In order to define what a regular lattice is, it is important to recall some basic notions on posets and lattices. It is necessary to use mathematical types for its better understanding. So, given a poset $(L, \leq)$ and $S, T \in L$, we have $S < T$ for $S \leq T$ and $S \neq T$. We write $S \lessdot T$ if $S < T$ and there is no $U \in L$ with $S < U < T$. In this case we say that T covers S. we recall moreover that a meet of $S, T \in L$ is a maximal lower bound for both S and T. Similarly, a join of $S, T \in L$ is a minimal upper bound for both S and T [59].

Therefore, a lattice is a poset $(L, \leq)$ where every $S, T \in L$ have a unique meet and a unique join, denoted by $S \wedge T$ and $S \vee T$, respectively.

Also, the meet and join of a lattice $Ł = (L, \leq, \wedge, \vee)$ define two binary, commutative and associative operations $\wedge, \vee : L \times L \rightarrow L$. Specifically, for any non-empty finite subset $M \subseteq L$, the lattice elements $\{S : S \in M\}$ and $\{S : S \in M\}$ are well defined. When $Ł$ is finite (i.e., L is finite), we set $0Ł := \{S : S \in L\}$ and $1Ł := \{S : S \in L\}$. A finite lattice $Ł$ is graded of rank r if all maximal chains (with respect to $\leq$) in Ł have the same length r. We denote the rank of a graded lattice $Ł$ by

rk(Ł). Thus, there exists a unique function ρŁ : L → {0,...,r}, called the rank function of Ł, with ρ(0Ł) = 0 and ρŁ(T) = ρŁ(S) + 1 whenever S $\lessdot$ T. The function ρŁ is monotonic, i.e., ρŁ(S) ≤ ρŁ(T) whenever S ≤ T . Moreover, ρŁ(L) = {0,...,r}, and 0Ł and 1Ł are the only elements of rank 0 and r, respectively.

So, finally we are able to give a definition of what a regular lattice is. It is about a finite graded lattice Ł = (L, ≤, ∧, ∨) of rank r, which depends on the two following conditions [59]:

(a) For all T ∈ L and for all integers 0 ≤ s ≤ r,

- the number of S ∈ L with ρŁ(S) = s and S ≤ T only depends on s and ρŁ(T),
- the number of S ∈ L with ρŁ(S) = s and T ≤ S only depends on s and ρŁ(T).

(b) For all S, T ∈ L with S ≤ T , the Möbius function μŁ(S, T) only depends on ρŁ(S)
and ρŁ(T) [59].

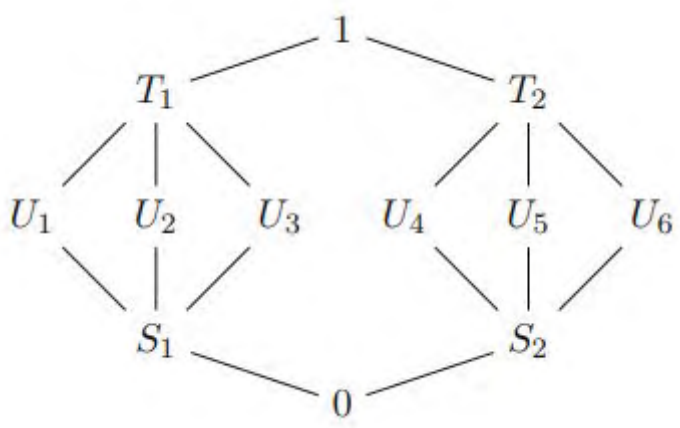


Figure 3.1: A regular lattice [59]

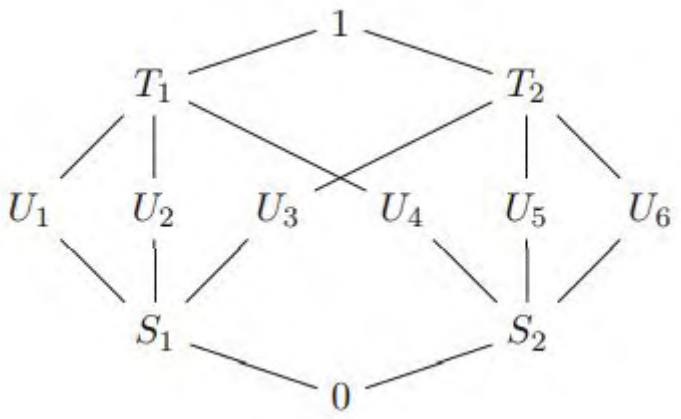


Figure 3.2: A non-regular lattice [59]

In Figures 3.1 and 3.2, the difference between a regular and non-regular lattice is depicted.

More specifically, in the field of networks, a regular lattice is a network that consists of n nodes, where each node has the same number and the same pattern of connections with every other node in the network. The degree distribution (in other words, the way nodes are connected to each other) of a regular lattice is uniform (constant).

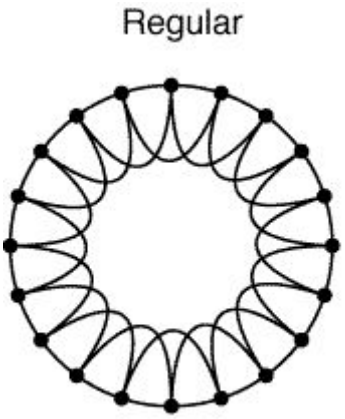


Figure 3.3: Illustration of a regular lattice [11]

Several models of networks have been proposed. A very simple and world-wide implementation in networks is based on the theory of two mathematicians, Erdos and Renyi (ER) who suggested, that the network is modeled by connecting its nodes with randomly placed links. An important prediction of random network

theory is that, despite its random construction, the resulting system will be deeply democratic [9]. In other words, most nodes will have approximately the same number of links. Since links are distributed in an uncorrelated way, degree distribution is Poissonian, which means the nodes follow a Poisson distribution, often known as the distribution of rare events, with a bell shape, as follows:

$$p(k) = e^{-\lambda} \cdot \frac{\lambda \cdot k}{k!}$$

where P(k) is the frequency of nodes with k links and λ is the average degree, λ =< k >, of the entire graph [14]. For this reason, random networks are also called exponential, because the probability that a node is connected to k other sites decreases exponentially for large k [9] and as a result, a homogeneous network is generated (low clustering) [3]. In conclusion, this means that it is extremely rare to find nodes that have significantly more or fewer links than the average [9].

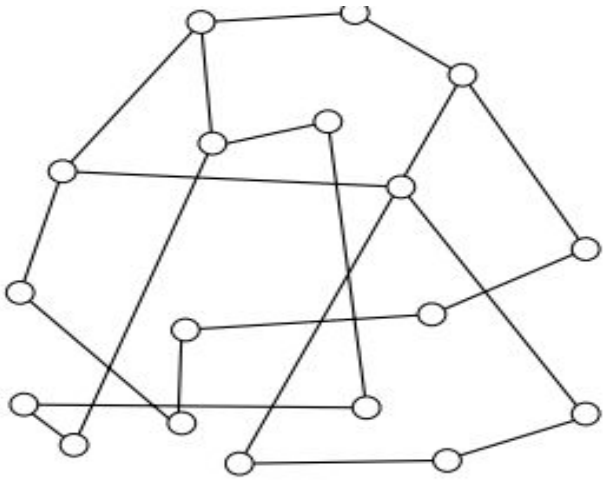


Figure 3.4: Classic Erdos-Renyi model [14]

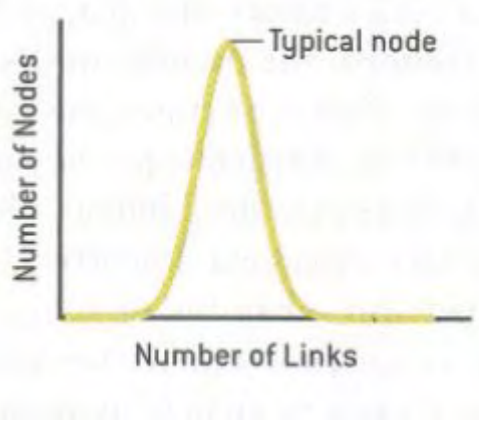


Figure 3.5: Bell Curve Distribution of Node Linkages [9]

However, this network has some serious shortcomings, taking in mind that this existing model fails to take into account important attributes of most real networks [16]. The most serious of all is its degree distribution. These models assume uniform probabilities when creating new edges, something that is not realistic [3]. As we understand, this kind of networks are pretty simple and unfortunately, this simplicity is a drawback as far as applications go. Furthermore, as we can see, real networks are open and they are dynamically formed by continuous addition of new nodes to the network [3]. A very important example of a real network is the internet. The WWW is continually sprouting new webpages and the research literature constantly grows since newspapers are continuously being published [3]. Therefore, this problem makes the random graph a poor approximation to the real-world networks [16]. So, it is important to find a more efficient degree distribution so that we can describe and solve problems of real networks [3].

It is known that both regular and random graphs are two exactly different types of networks. The first one is a network which has the lowest heterogeneity (meaning that the number of connections each node has is approximately the same) and lowest randomness, concerning the links between the nodes. In other words, in regular graphs, nodes tend to be densely connected in groups (long average path and high clustering). On the contrary, in random ER (Erdos-Renyi) graphs, most nodes have the same number of connections (low heterogeneity), but the degree distribution is a Gaussian bell-shaped curve, as it is described in paragraph, referred to as *Random Network*, in this section. So, Random graphs (constructed by ER method) have short average path and low clustering [34]. However, taking into consideration the needs of the "real-world" networks, (as neuronal networks, food webs, social networks, scientific-collaboration networks, computer networks and so on) neither random networks, nor regular lattices seem to be an adequate framework within which scientists can study more complex networks [35]. In 1998, in order to describe the transition from a regular lattice to a random graph, Watts and Strogatz (WS) introduced the concept of small-world network [3].

```
1: Initialization:
2:       Set the total number of nodes N
3:       Set the rewiring probability p
4:       Set the node degree K
5: end initialization
6:
7: for each node n ∈ (1, N)
8:       Connect to nearest K nodes.
9: end for
10:
11: for each node n ∈ (1, N)
12:      for each link k ∈ (1, K)
13:           Generate a random number r ∈ (0, 1)
14:           if r < p
15:              Select a node randomly m ∈ (1, N)
16:              Rewire to node m
17:           end if
18:      end for
19: end for
```

Figure 3.6: Small-world network algorithm [12]

It is a model, where the connections between the nodes in a regular graph are rewired with a certain probability, following the Poisson degree distribution. Specifically, the typical distance between two randomly chosen nodes grows proportionally to the logarithm of the number of nodes in the network. Watts and Strogatz proposed a model which has a higher clustering and almost the same average path than the random networks with the same number of nodes and edges [34]. More specifically, a small-world graph is created, based on a regular lattice and each node in the network is connected to K nearest nodes. To construct a small world network, it is necessary to use a rewiring probability p with $0 <= p <= 1$. For every link of each node in the graph, we generate a random number r with $0<r<1$. In case of number r is smaller than probability p, a node m is selected randomly and the current link is rewired to the node m. By controlling the rewiring probability p, the network will interpolate between a regular lattice ($p = 0$) to a random network ($p = 1$) [7] [12]. In Figure 3.6, we see the small-world algorithm in a pseudocode format.

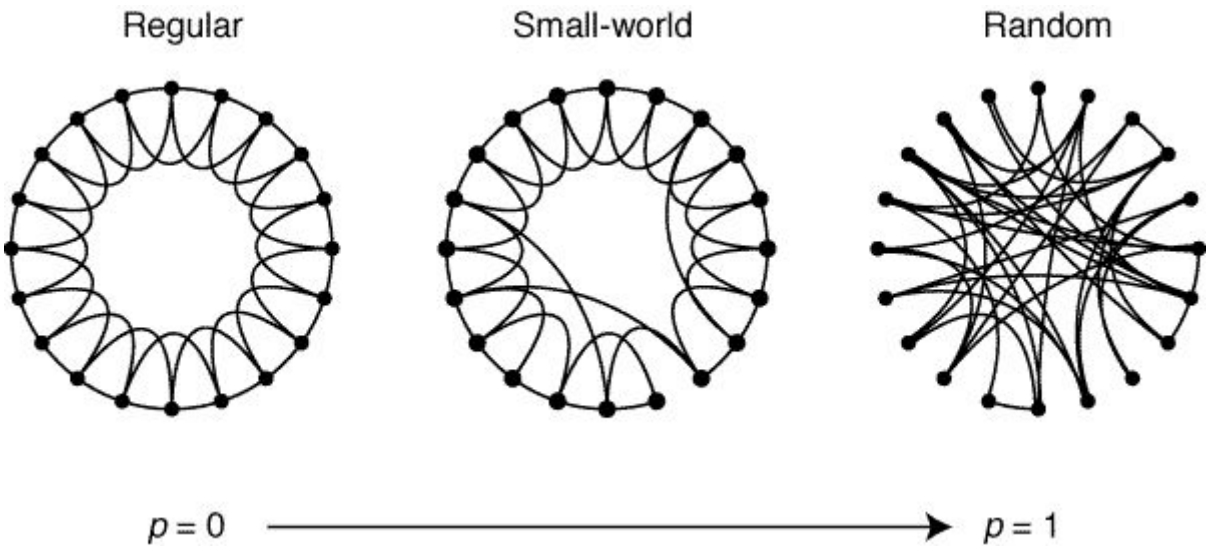


Figure 3.7: Increasing randomness [11]

Furthermore, Figure 3.7 shows how it is possible, by increasing the rewiring probability, to remodel a regular net to small-world one by rewiring the links. Comparing the connections in graphs, we conclude that small-world networks has random connections but not as many as random graphs. Mathematically, the first property of small world networks (high clustering coefficient) is given by the following type:

$$C = \frac{2e}{k(k-1)}$$

where e is edges between the neighbors of a node, k is the degree of the node, so k(k-1) is the total number of possible edges between neighbors. High clustering coefficient shows that nodes with high degree easily share informations with other nodes. The second property (small average path length) is the distance between nodes in the graph and it is given by this mathematical type:

$$L = \sum_{ifn,i \neq j} d_{ij} \cdot \frac{1}{N(N-1)}$$

where d_{ij} is the e shortest geodesic distance between nodes i and j. So path length is calculated as the average of the shortests paths between all possible node pairs. If the parameter L takes small values, then information can easily be distributed across the network.

Specifically, in Figures 3.8, 3.9 and 3.10, we see how the growing rewiring probability affects the network.

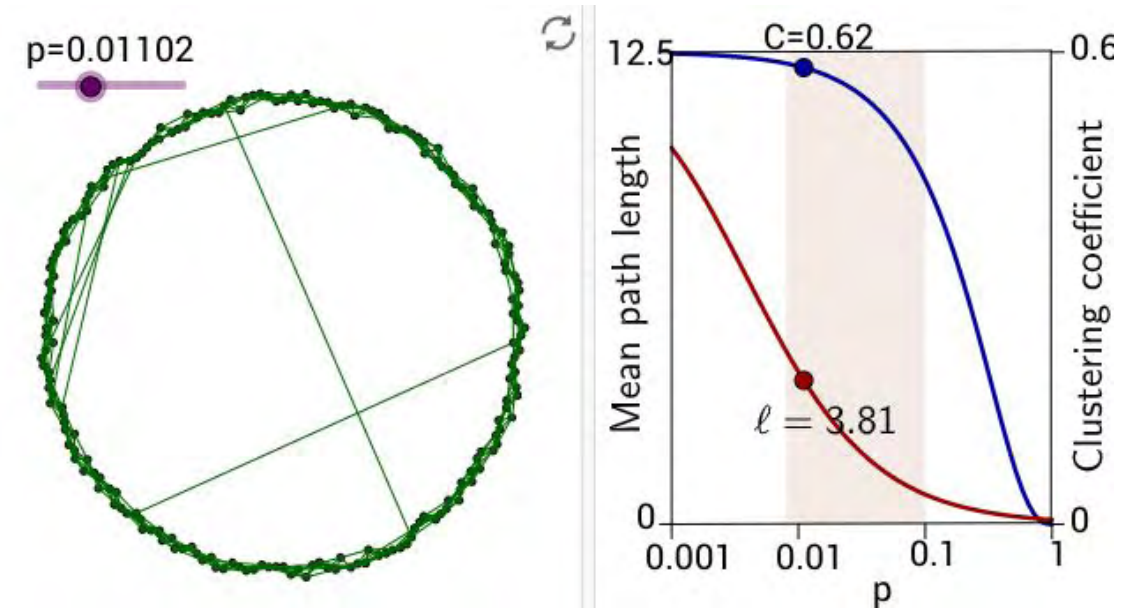


Figure 3.8: Network construction, using Watts and Strogatz's small-world model with p=0.01102 [60]

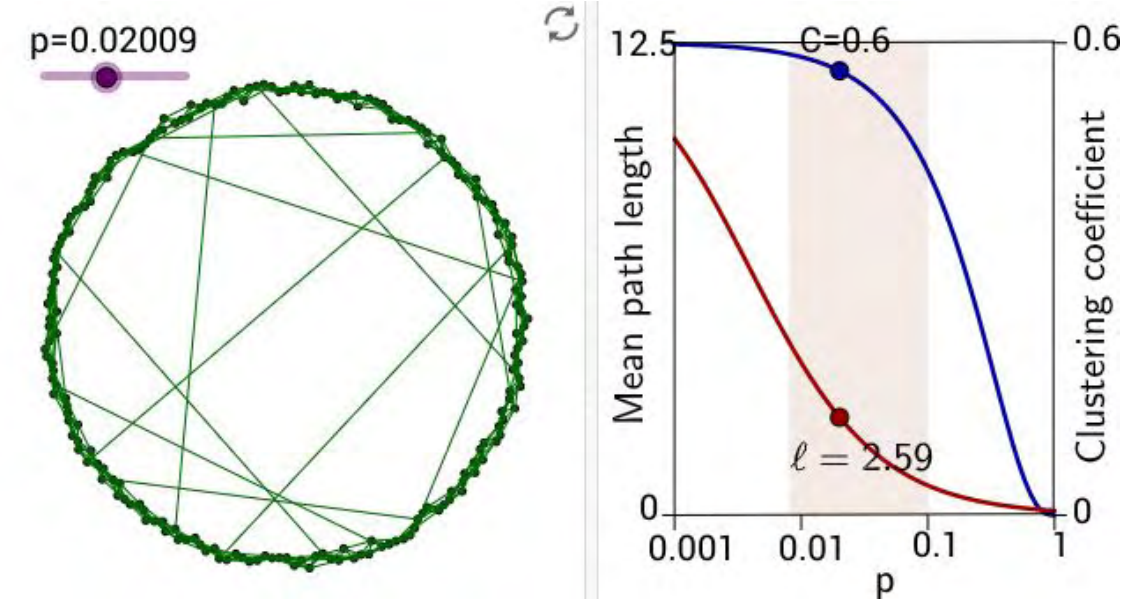


Figure 3.9: Network construction, using Watts and Strogatz's small-world model with p=0.02009 [60]

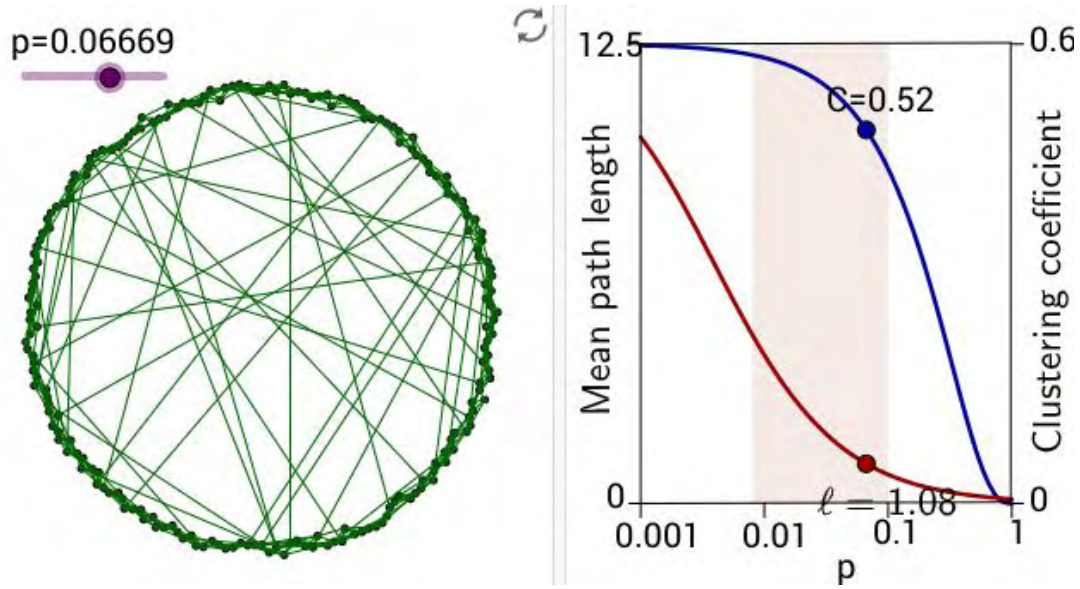


Figure 3.10: Network construction ,using Watts and Strogatz's small-world model with p=0.06669 [60]

Figures 3.8, 3.9 and 3.10 show that in small-world model, path length decreases abruptly while clustering decreases smoothly. Blue line represents high clustering and red line represents the short path length. The rewiring probability p has values in range 0 to 1 and while p increases, the network become denser and tends to get attributes that belong to random graphs. In cases of p, having small values, the network has high clustering and small path length [10].

As mentioned above, small-world is about a model which has clustering close to that of a lattice and path lengths similar to those of random networks. Although small-world networks are an improved method of describing complex networks, the "real-world" networks are not homogeneous (each node has about the same number of link connections [3]) ones, meaning that we are in need of a more strictly constructed graph, which can describe these phenomena,too.

Over the past two decades, networks of complex topology have been described with the random graph theory of Erdős and Rényi (ER) [70]. As mentioned above, the Erdos-Renyi network is a random graph obtained by randomly distributing M links between N nodes, being a statistical ensemble with equal probability for any generated configuration [14].

In the past few years, due to the absence of data on large networks, the predictions of the ER theory were rarely tested in the real world [70]. Later, many empirical results showed that for most large-scale real networks the degree

distribution deviates significantly from the Poisson distribution. Specifically, for a large number of networks, the degree distribution can be better described by a power law, whose form is $p(k) \sim ck^{-\gamma}$, where c is a corresponding positive constant for predetermined N and γ is some exponent which satisfying $\sum_{k=1}^{N} p(k) = 1$ [26]. This power-law distribution falls off more gradually than an exponential one and allows a few nodes of very large degree to exist. In addition, we want to make a model of a large network for which we know the degree distribution but nothing else [16], so as this network can be dynamically evolved. To explain the origin of power-law degree distribution, Barabási and Albert (BA) proposed another network model, known as scale-free network [3].

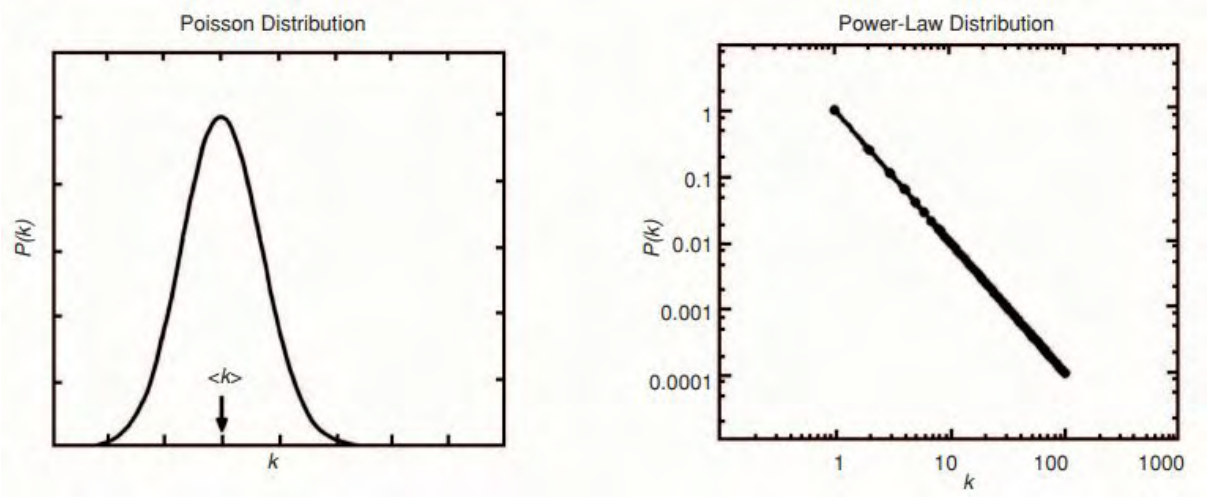


Figure 3.11: Poisson Distribution vs. Power-Law Distribution for k nodes [3]

A scale-free network is a network ,whose degree distribution follows a power law [25]. This network was grown under the preferential attachment rule. The network starts the evolution process with a small number of nodes [12]. Then, at each iteration a new node is added to the network and connected to m already existing nodes with a probability of linking to a certain node proportional to the actual degree (number of links) of that node [14]. In other words,nodes are added to the network with a preferential bias toward attachment to nodes which already have a high degree [26].

Examples of complex networks, whose vertex connectivities follow a scale-free power-law are systems such as genetic networks or the World Wide Web [70].

In Figure 3.12, we can see a pseudocode format, which modelizes this type of network.

1: **Initialization:**
2: *Set* the initial number of nodes m_0
3: *Set* the total number of nodes N
4: *Set* the node degree K
5: **end initialization**
6:
7: **for** each node $n \in (1, m_0)$
8: *Connect* to nearest K nodes.
9: **end for**
10:
11: **for** each new node
12: *Calculate* the maximum degree probability as follows:

$$\prod(k(h)) = k(h) / \sum_m k(m),$$

where $\Pi(k(h))$ is the probability of selecting node h, $k(h)$ is the degree of node h, and $\sum_m k(m)$ is the total number of links in the network.

13: **for** each link $k \in (1, K)$
14: *Connect to* node m with $\max(\Pi(k(m)))$
15: **end for**
16: **end for**

Figure 3.12: Scale-Free network algorithm [12]

Thus, inspired by the Network Science theory, in this work, we describe five new algorithms, based on SET (described in the *SET algorithm* paragraph, in this section) in which we implement scale-free and small-world techniques, in order to see whether topology plays a significant role in training process acceleration and how this idea is going to affect the network accuracy. Our goal is to speed up the training time, without sacrificing the accuracy.

So, in SET code, a MLP (Multi-Layer Perceptron) neural network is used and trained, given some data as input, in order for the neural network to make future predictions about whether a person is sick or will get sick. It is introduced a procedure which takes into consideration data distributions and creates sparse bipartite layers suitable to replace the fully-connected bipartite layers in any type of networks [1]. Its construction is based on attributes of random graphs. Its nodes are linked randomly. Set algorithm aims to accelerate the training of this neural network by sparsing its topology, using a sparse table to represent the links between the nodes. In fact, it starts by creating a random graph and training it, using back propagation method as it is described in section 3.1.2. During the training, set algorithm implements a method which sparses the topology of the existing random graph, in each epoch except the last one. The sparsity of the network is achieved by removing the weights with values close to zero in each epoch, as these links don't affect the network. Then, in order to maintain the balance on the network, we need to introduce as many links to the system as we have removed, giving them random weights. In this particular algorithm this process is done randomly. In conclusion, it starts from an Erdős–Rényi random graph topology and throughout training process, network ends up with a more structured connectivity, like scale-free topology [1].

In Figure 4.1, we make a short introduce to the Set algorithm.

Algorithm 1: SET pseudocode

```
%Initialization;
initialize ANN model;
set ε and ζ;
for each bipartite fully-connected (FC) layer of the ANN do
    replace FC with a Sparse Connected (SC) layer having a Erdős-Rényi topology given by ε and Eq.1;
end
initialize training algorithm parameters;
%Training;
for each training epoch e do
    perform standard training procedure;
    perform weights update;
    for each bipartite SC layer of the ANN do
        remove a fraction ζ of the smallest positive weights;
        remove a fraction ζ of the largest negative weights;
        if e is not the last training epoch then
            add randomly new weights (connections) in the same amount as the ones removed previously;
        end
    end
end
```

Figure 4.1: (Decebal Constantin Mocanu etc, 2018) SET pseudocode [1]

In order for the neural network to be trained, a learning algorithm is trained on a set of data, and then the model is applied to make predictions on new data points. The goal is to maximize its predictive accuracy on the new data points. Specifically, we have to represent the data in a specific form, so as for the model to avoid fitting the noise in the data by memorizing various peculiarities of the training data rather, than finding a general predictive rule. In other words, we want to avoid the phenomenon of overfitting [17]. The existence of this phenomenon leads the model to have a low accuracy. In order to prevent this case from happening, we need to regularize the data. Regularization is a form of regression, that regularizes or shrinks the coefficient estimates towards zero. So, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. In this project, the datasets we use, contains biological data which are data of life sciences information, collected from scientific experiments, published literature, high-throughput experiment technology, and computational analysis, so that the neural network can make predictions about whether a patient will become sick, concerning its symptoms. Also, we use a fifth dataset with images, which means that algorithms through neural networks, try to predict what these images are depicted. The method used for regularization is one-hot encoding, which creates new (binary) columns, indicating the presence of each possible value from the original data.

In this project we implemented five algorithms in python which tend to speed up the training of neural network by sparsing the topology. These algorithms implementation are based on a MLP neural networks and, during the training phase, back propagation is used for weights update. In all cases, we start with a specific type of network (scale-free or small-world) and through the training procedure, we sparse this network by removing and reconnecting links, using the techniques of scale-free and small-world graphs, in each epoch except the last one. So, in the end we have a new neural network which is sparsed. The training procedure is described in section 4.1. In the sections of this chapter that follow, we describe our detailed experimental work.

In this algorithm, we start by implementing an exact scale-free network [9] and we end up by creating a sparsed network, similar to scale-free one, based on the SET code. Firstly, we create a scale-free graph by creating a sparse table, representing the connections between the nodes in each layer. Then, we remove the weights close to zero (sparsity) and add as many links as we removed to the most powerful node, using the scale-free method. If a link, having to be reconnected, already exists, then changes don't happen. In order to find the most powerful node, we calculate the probability of each node, which is defined as the quotient of the incoming connections of this node regarding all of the graph connections. Every link is reconnected to the node, which probability is bigger, and the new weight of the link is random. By this procedure, we create a scale-free network which is trained using back propagation method as it is described in section 3.1.4. We end up in a sparsed graph similar to scale-free, using the SET algorithm, which during the training, removes the links close to zero and adds as many links as removed, in a random way, after each epoch except the last one. So ,the difference between scale-free and set algorithm is in the part where links are reconnected. In first one, links are reconnected to the node with the maximum degree probability, following power law distribution, while in the second one, links are reconnected randomly.

Pseudocode format of Scale Free to SET is given in Algorithm 1.

Scale Free to Set

1. initialize a sparse table randomly.
2. remove links ,whose weights are close to zero.

3.
4. each node i of every layer:
5. calculate the maximum degree probability as follows:
6. $p_i = \Sigma_{\text{incoming links (i)}} / \Sigma_{\text{links in network}}$
7.
8.reconnect the nodes,whose link was removed, with the node that has the maximum p.
9. the weight of the new connection is given, randomly.
10.
11. this link exists:
12. nothing is done.
13.
14.
15. each epoch:
16. remove links, whose weights are close to zero.
17. reconnect the nodes, whose link was removed, randomly.
18. the weight of the new connection is given, randomly.
19.

For this algorithm, complexity is calculated in detail, in respect with the code. For the rest of the proposed algorithms, computational complexity is produced, bearing in mind only the repetitive code instructions (for, while), due to the fact that the running time of the algorithm is proportional to the number of loops. The running time of a simple statement is constant (O(1)). Furthermore, the initialization cost is the same for all the variants, proposed in this project and is equal to: 13 variables initialization + 4 loop instructions for every layer (L) + 10 variables initialization (including table initializations) for every layer (L) = $13 + (4+10) * L = 13 + 14 * L = O(L)$. Specifically, the 'remove', the 'Scale Free' and the 'SET' parts of the code are analyzed, here.

Remove connections complexity:
16 instructions (including value assignments and calculations) + (5 + 10) loop instructions (including value assignments, function calls and comparisons between variable values) instructions for every layer (L) + 2*3 instructions (value assignments) + (3 + 4*2 + 4 + 7 + 2 +3) instructions for every layer (L) + 4 instructions for every node (N) * 3 instructions for every connection (C) in every node (N) * 6 instruction for every connection (C)= $16 + 15 * L + 6 + 27 * L + 72 * N * C * L + 22 + 42 * L + 72 * N * C * L * C * L$
$= O(N * L * C)$.

Scale Free part complexity:
3 instructions (2 calculations + 1 comparison) + (2 loop instructions + 16 other instructions) for every layer (L) * (2 loop instructions + 11 other instructions) for every node (N) = 3 + (13*N + 18)*L = $O(N * L)$.

The 'Scale Free' part is executed one time when the code starts. On the contrary, the 'remove' part is repeated after each epoch, combined with the formula that reconstructs the initial network, giving it same or different attributes.

In Scale Free to SET algorithm, the reconstructed network follows SET method.
SET complexity:
(2 loop instructions + 63 other instruction) for every layer (L) * (1 instruction per loop iteration + 23 other instructions) for every connection that is removed (R) + 8 other instructions = (2 + 63 + (23 + 1) * R) * L + 8 = $O(L * R)$.

According to these results, we conclude that the most time-expensive part is the one that sparsifies the network.

As mentioned in section 4.1 SET method uses an network, which starts from a random sparse topology (Erdös–Rényi random), evolving through a random process during the training phase towards a scale-free topology. Remarkably, this process does not have to incorporate any constraints to force the scale-free topology. In other words, evolutionary algorithm is not arbitrary, which means it follows a phenomenon that takes place in real-world complex networks (such as biological neural networks and protein interaction networks) [1]. On the contrary, in our algorithm we implement an exact scale-free topology, which means that links are reconnected to the nodes, following power low distribution. We start by creating an exact scale-free network, using the scale-free method as it is described in section 3. In the part of the code, where network is sparsed, we remove the weights close to zero and add as many links as we removed to the most powerful node, using again the scale-free method as it is described in section 3.1.4.

Scale Free to Scale Free

1. initialize a sparse table randomly.
2. remove links, whose weights are close to zero.
3.
4. each node i of every layer:
5. calculate the maximum degree

probability as follows:

6. $p_i = \Sigma_{\text{incoming links (i)}} / \Sigma_{\text{links in network}}$
7.
8. reconnect the nodes, whose link was removed,
   with the node that has the maximum p
9. the weight of the new connection is given,
   randomly.
10.
11. this link exists:
12. nothing is done.
13.
14.
15. each epoch:
16. each layer:
17. do steps 2 to 13.
18.
19.

Concerning the computational complexity, this algorithm starts with a scale free network (O(*N*L*)) and via randomization (O(*N*L*C*)), ends up in a same-attributed network (O(*N*L*)), as analyzed in section 4.3.1.

In this particular algorithm we start with a scale-free implementation, and we end up in a similar type of network, using an alternative version of scale-free technique. Particularly, in every epoch, except last one, we remove the links with weight close to zero and add as many links as we removed (sparsity) to the most powerful node (node with the greatest probability). In original method, if a link, from one node,in a layer, to another, which has to be reconnected, already exists, then changes don't happen. So in this version, if the link we want to add to the most powerful node already exists then we try to add a connection to the second most powerful node and so on till the fifth strongest node. Both in the original version and in our alternative one, the weights are randomly given. We make this variant of Scale-Free to Scale-Free algorithm in order to add as many links as we can. If a link we try to reconnect already exists, we try to reconnect it to the successive (regarding the degree probability) node and so on, trying to maintain the balance between the removed links and the ones that have to be reconnected, in the network.

Scale Free to Scale Free (5)

1. initialize a sparse table randomly.
2. remove links ,whose weights are close to zero.
3.
4. each node i of every layer:
5. calculate the maximum degree probability as follows:
6. $p_i = \Sigma_{\text{incoming links (i)}} / \Sigma_{\text{links in network}}$
7.
8. reconnect the nodes, whose link was removed, with the node that has the maximum p_i
9. the weight of the new connection is given, randomly.
10.
11. this link exists:
12. nothing is done.
13.
14.
15. each epoch:
16. each layer:
17. each node:
18. do steps 4 to 7.
19. reconnect all links removed,as follows:
20. j in 5 strongest nodes of next layer:
21. the link in j node doesn't exist:
22. connect node to j
23.
24.
25.
26.
27.
28.

This algorithm starts from a scale-free structured network, whose complexity is O(*L*N*) and ends up through randomization (O(*L*N*C)*) (analyzed in section 4.3.1) in a same-attributed network which differs only, in the part of dominant nodes (in this case we have 5 popular nodes instead of one). Here, the number of iterations that find the 5 strongest nodes is considered as stable regardless the dataset and also it is a very small number proportionately to the N and L. Thus, the complexity of the final net is 5**L*N* = O(*L*N*).

Here, we start implementing a scale-free network and after the procedure of training, we construct a small-world type of network. In the first place, after removing connections with no important impact, we calculate the degree probability of every node and then reconnect these nodes (in every layer), whose link was deleted,to the most powerful node of the next layer. In section 3.1.4 there are more details about scale-free technique. After the training part of the algorithm we proposed, a small-world network is created. Specifically, a rewiring probability is defined. In order to construct the network, we chose small values of this probability (p = 0.02 and p = 0.075), because the smaller the probability is, the less density the network has (sparsity). For each node in every layer, we find its links, whose weight is non-zero and give them a random probability. After that, links whose probability is smaller than rewiring probability, are disconnected and, then, we try to rewire them in a random node, giving them a random weight value (only if the connection to this randomly chosen node, doesn't exist, else we try to find another random node to connect to). In this experiment, we want to see how much the performance (not only the accuracy, but also the training time) of the network is affected when we start from a strictly structured network and through the process of training, we create a network with a more arbitrary structure.

Scale Free to Small World

1. initialize a sparse table randomly.
2. remove links ,whose weights are close to zero.
3.
4. each node i of every layer:
5. calculate the maximum degree probability as follows:
6. $p_i = \Sigma_{\text{incoming links (i)}} / \Sigma_{\text{links in network}}$.
7.
8. reconnect the nodes, whose link was removed, with the node that has the maximum p_i
9. the weight of the new connection is given, randomly.
10.
11. this link exists:
12. nothing is done.
13.
14.
15. each epoch:

16. define a probability.
17. each layer:
18. each node n in this layer:
19. find links, whose weight is non-zero.
20. give those links a random probability P_{link}.
21. find N of those links, so as: $P_{link} < P$.
22. each N:
23. select a node m in next layer randomly, so that there is no connection between node m and current node n.
24. rewire current node n to node m.
25.
26.
27.
28.

Firstly, the initial network complexity is O(L*N) as mentioned in 4.3.1. The final network this algorithm creates is more complicated. Especially, the removing part takes place during the reconstruction of network instead of the beginning of the network as it is in scale-free implementations. Specifically, the complexity of small-world algorithm is: For every link (P) in every node (N) of each layer (L), we find random nodes in next layer (N') in order to rewire a link in P, whose probability is smaller than the rewiring one. So, complexity is O(*L*N*P*N'*).

In the last algorithm, we implement a network, based on small-world technique and through the process of training, a same type of network is constructed. It is interesting to see how the transition, being from a less randomly constructed network (small-world) to another, affects the network performance. These small-world networks are created in the same way as it is described in section 3.1.3. We introduce a detailed pseudocode format in algorithm 5.

Small World to Small World

1. define a probability.
2. each layer:
3. each node in this layer:
4. find links, whose weight is non-zero.
5. give those links a random probability P_{node}.
6. find numbers N of nodes , so as: $P_{node} < P$.
7. each N:

8. select a node m in next layer randomly ,so that its link weight
9. is zero.
10.
11. rewire current node to node m.
12.
13.
14.
15.
16. each epoch:
17. do steps 2 to 14.
18.
19.

The computational complexity of small-world algorithm is the same with the one analyzed in section 4.3.4. In this case, the initial and the final net are of the same type, except for the initialization part in the initial net which has complexity O(*L*). Therefore, the final net complexity is O(*L*N*P*N'*).

This section presents details about the dataset used and about the size of the neural network we experiment with; then it gives the results of the experimental evaluation of the developed algorithms.

In order to test the algorithms, we used datasets which have hundred instances and few thousands features, as described in Table 5.1, in this section.

By the word *instances*, we define the number of input vectors, which are composed of as many components as features are, given to the neural network in order to be trained and then, evaluated. Every dataset has, also, different number of classes, which are the groups in which data are separated.

lung	203	3312	5
lung_discrete	73	325	7
TOX_171	171	5748	4
CLL_SUB_111	111	11340	3
COIL20	1440	1024	20

Table 5.1: Datasets characteristics.

Lung - Lung_discrete

This data was used by Hong and Young to illustrate the power of the optimal discriminant plane even in ill-posed settings. Applying the KNN method in the

resulting plane gave 77% accuracy. However, these results are strongly biased.The data described 3 types of pathological lung cancers. The Authors give no information on the individual variables nor on where the data was originally used [42].

TOX-171

This database is an example of the use of toxicology to integrate diverse biological data, such as clinical chemistry, expression, and other types of data. The database contains the profiles resulting from the three toxicants: alpha-naphthyl-isothiocyanate, dimethylnitrosamine, and N-methylformamide administered to rats. The classification task is to identify whether the samples are toxic, non toxic or control. Sample is toxic if alpha-naphthylisothiocyanate, or dimethylnitrosamine or n-methylformamide is administered, non-toxic if caerulein or dinitrophenol or rosiglitazone is administered and control if untreated [33].

CLL-SUB-111

The database has gene expressions from high density oligonucleotide arrays containing genetically and clinically distinct subgroups of B-cell chronic lymphocytic leukemia (B-CLL). The dataset is formed of 11340 attributes and 111 instances, as referred in Table 5.1 [33].

COIL20

This dataset contains 20 objects (number of classes). The images of each objects were taken 5 degrees apart as the object is rotated on a turntable and each object has 72 images. The size of each image is 32x32 pixels, with 256 grey levels per pixel. Thus, each image is represented by a 1024-dimensional vector.

As we mentioned in section 3.2, the datasets are encoded with one-hot encoding. Furthermore, every dataset is split in two parts, the training and the testing set. The ⅔ of the total size of dataset is used in order for the neural network to be trained and the ⅓, remaining, is used for testing its ability to learn the training set.

All the datasets were applied to all our proposed neural topology evolution algorithms except the variants Scale Free to Small World and Small World to Small World, in which the neural network takes enormous (much more than expected) time in order to be trained, in cases of using the large datasets (large number of features). Also, the accuracy is not satisfying. So, these two algorithms were trained and tested only with lung.mat and COIL20.mat files.

In all cases, a MLP (Multi-layer perceptron) neural network model is used. More specifically, it is about a neural network with an input level, three hidden layers and one output level. Each hidden layer has 1000 neurons and the number of neurons at the input and output levels depends on the dataset features and dataset classes, respectively. In each neural, we used activation function to produce output from each level which is given as input to the next layer. In our project, we chose the ReLU or FReLU functions for the hidden levels and the sigmoid function for the final level. In addition, the parameters epsilon with value 20 and zeta with value 0.3 were used. All algorithms functioned repetitively for 500 epochs, and the error at each time was calculated according to the mean squares error method. For the learning rate and batch size parameters, values of 0.01 and 2 were used respectively. In some experiments we use the momentum parameter with a value of 0.9. Finally, in cases where we implement regularization techniques, the weight decay parameter was set to 0.0002 for L2 regularization and 0.0000001 for L1 regularization method. In cases, we don't use regularization techniques (NoL), the weight decay parameter isn't taken into account.

Our software was tested on Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, installing operating system Ubuntu 16.04.1, with Python 3.5.2, Numpy 1.15.3, SciPy 1.1.0 and (optionally) Cython 0.29.

Taking into consideration the parameters, analyzed in paragraph 5.1.2, we run algorithms, described in paragraph 4.3, using five datasets with different number of samples and features, as mentioned in paragraph 5.1.1. By using these parameters, in this chapter we evaluate the performance of our algorithms.

In this section, we give the statistics (accuracy and training time) of the prior work that investigated a network sparsification approach [1], in order for our results to be compared with.

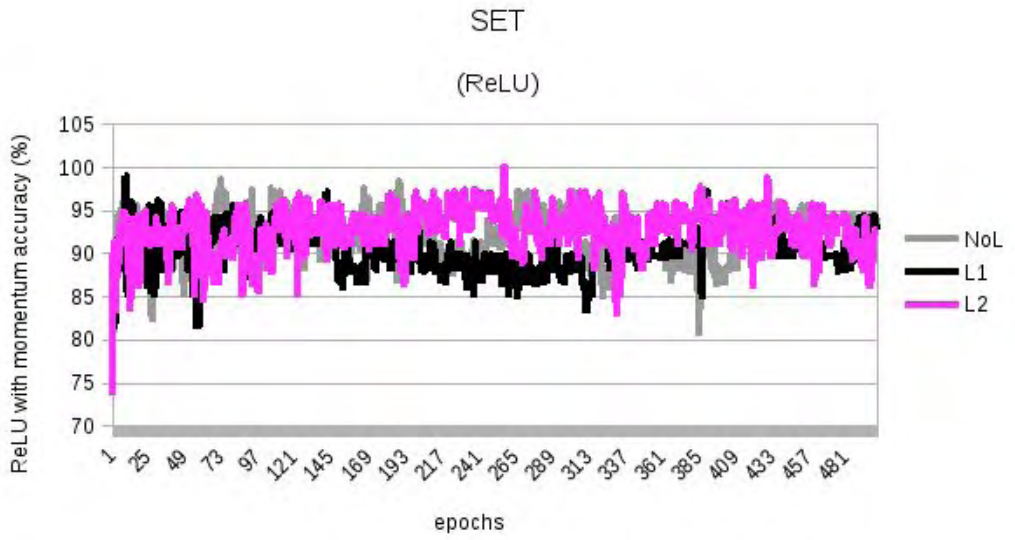


Figure 5.1: SET accuracy, using ReLU activation function and lung.mat file.

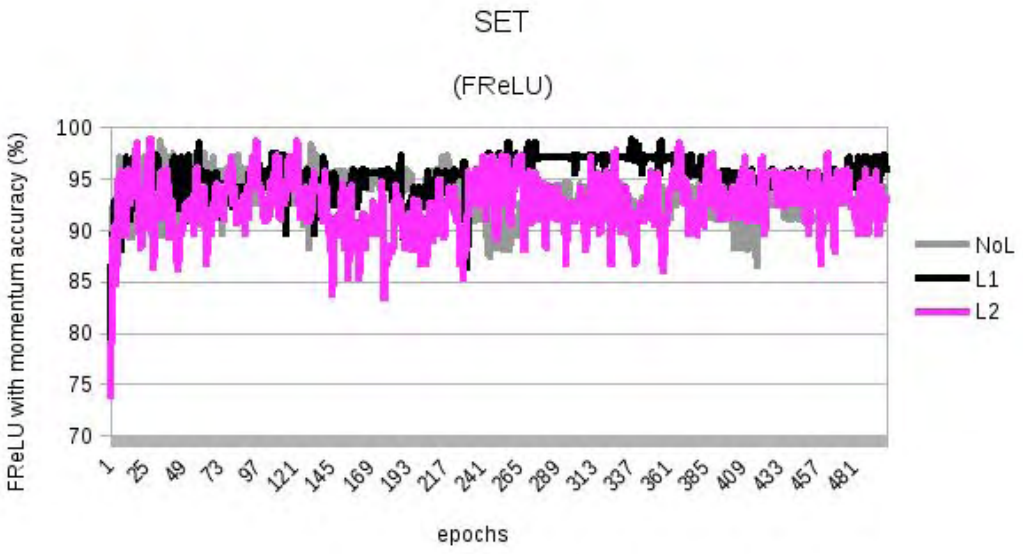


Figure 5.2: SET accuracy, using FReLU activation function and lung.mat file.

ReLU	NoL	92.02	35 mins
ReLU	L2	92.71	38 mins
ReLU	L1	90.53	47 mins
FReLU	NoL	93.02	45 mins

FReLU	L2	92.38	49 mins
FReLU	L1	95.04	53 mins

Table 5.2: Statistics of SET algorithm using lung.mat file.

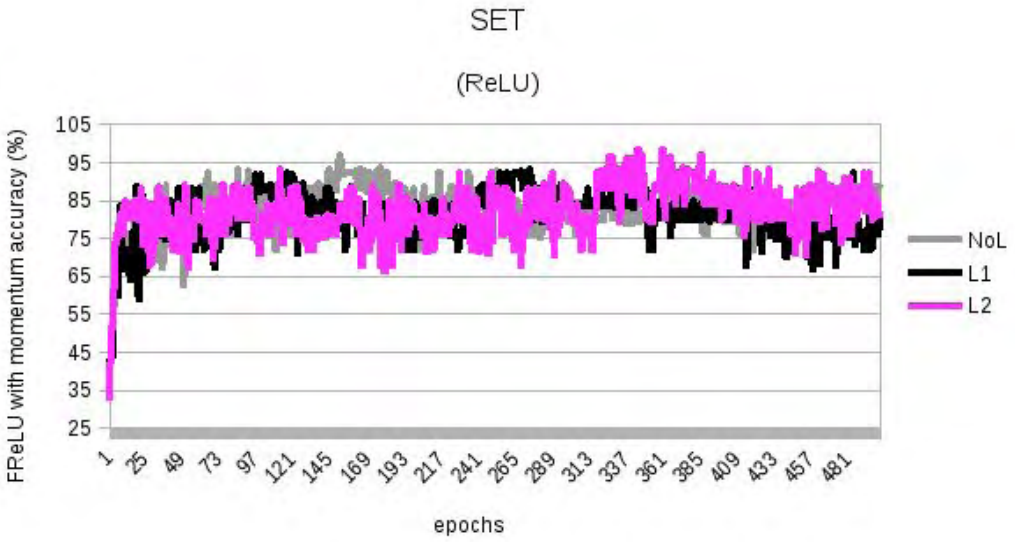


Figure 5.3: SET accuracy, using ReLU activation function and lung_discrete.mat file.

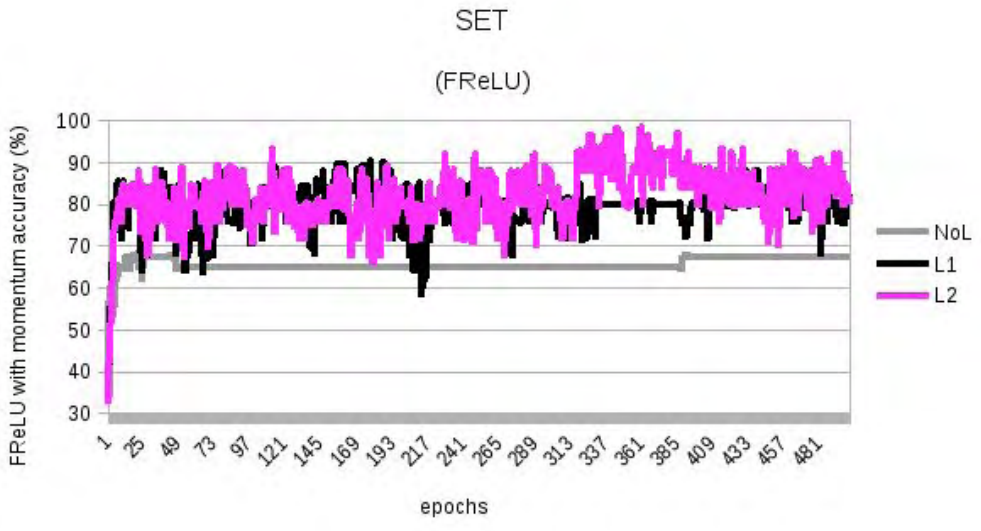


Figure 5.4: SET accuracy, using FReLU activation function and lung_discrete.mat file.

ReLU	NoL	82.59	11 mins
ReLU	L2	82.1	11 mins
ReLU	L1	80.9	13 mins
FReLU	NoL	65.5	13 mins
FReLU	L2	82.11	11 mins
FReLU	L1	79.62	12 mins

Table 5.3: Statistics of SET algorithm using lung_discrete.mat file.

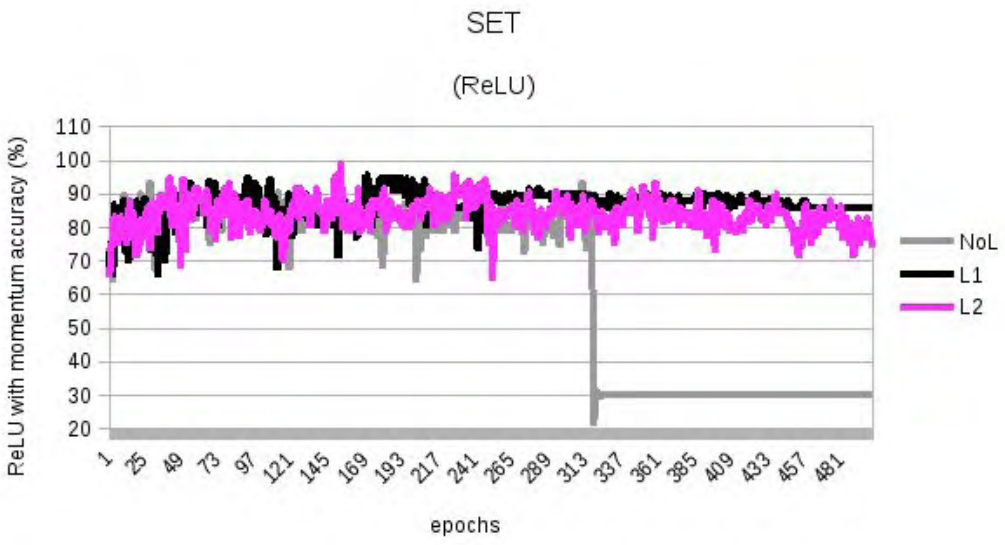


Figure 5.5: SET accuracy, using ReLU activation function and TOX_171.mat file.

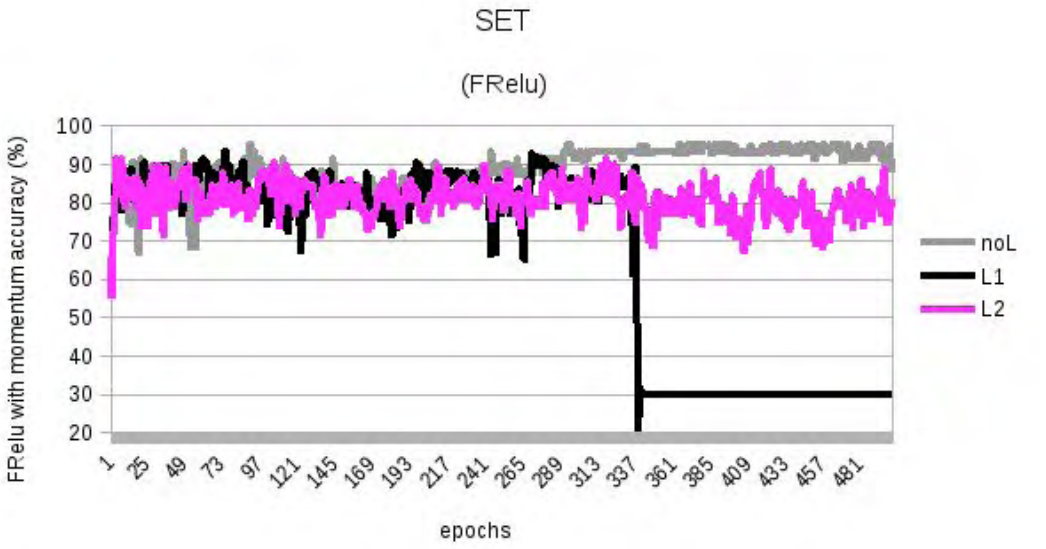


Figure 5.6: SET accuracy, using FReLU activation function and TOX_171.mat file.

ReLU	NoL	63.49	23 mins
ReLU	L2	83.69	27 mins
ReLU	L1	86.51	35 mins
FReLU	NoL	88.43	32 mins
FReLU	L2	80.9	33 mins
FReLU	L1	65.74	36 mins

Table 5.4: Statistics of SET algorithm using TOX_171.mat file.

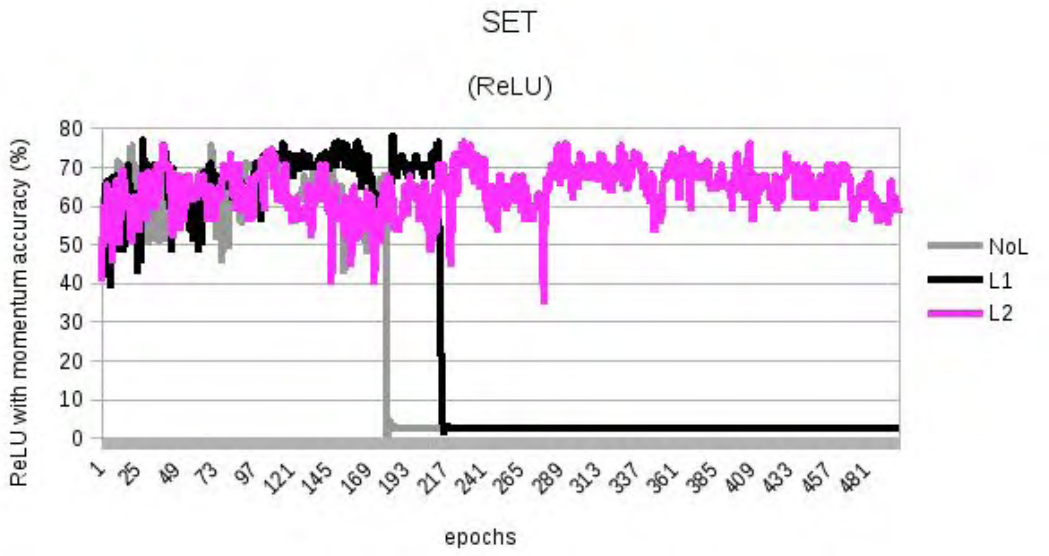


Figure 5.7: SET accuracy, using ReLU activation function and CLL_SUB_111.mat file.

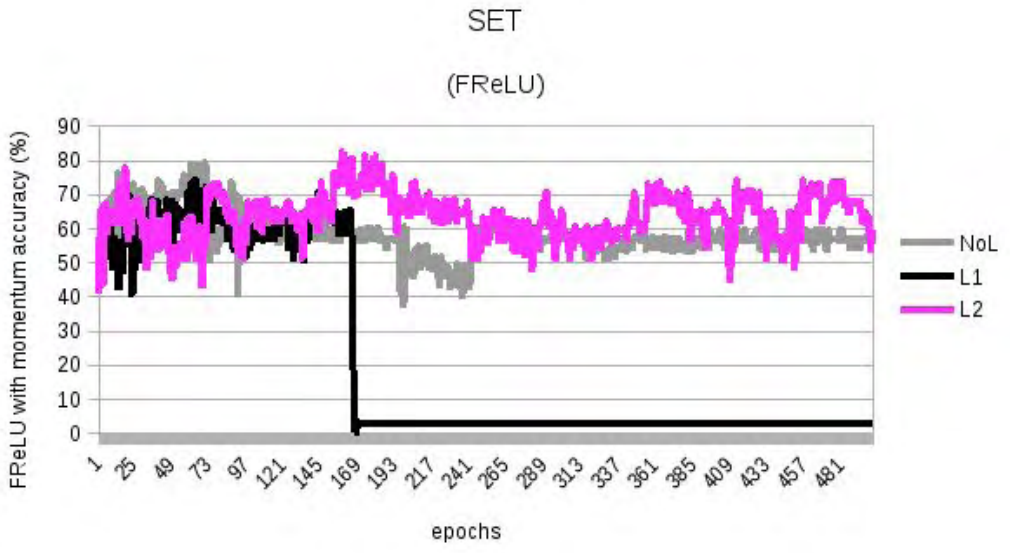


Figure 5.8: SET accuracy, using FReLU activation function and CLL_SUB_111.mat file.

ReLU	NoL	23.98	45 mins
ReLU	L2	63.84	45 mins
ReLU	L1	29.78	55 mins

FReLU	NoL	58.15	56 mins
FReLU	L2	63.54	57 mins
FReLU	L1	21.75	58 mins

Table 5.5: Statistics of SET algorithm using CLL_SUB_111.mat file.

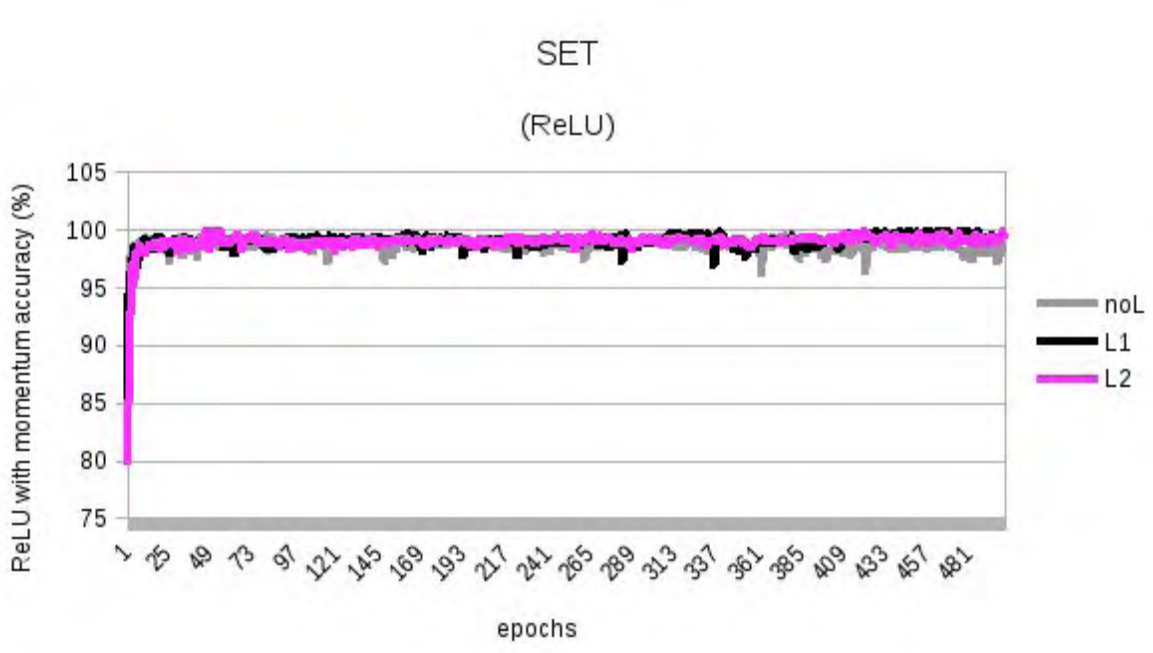


Figure 5.9: SET accuracy, using ReLU activation function and COIL20.mat file.

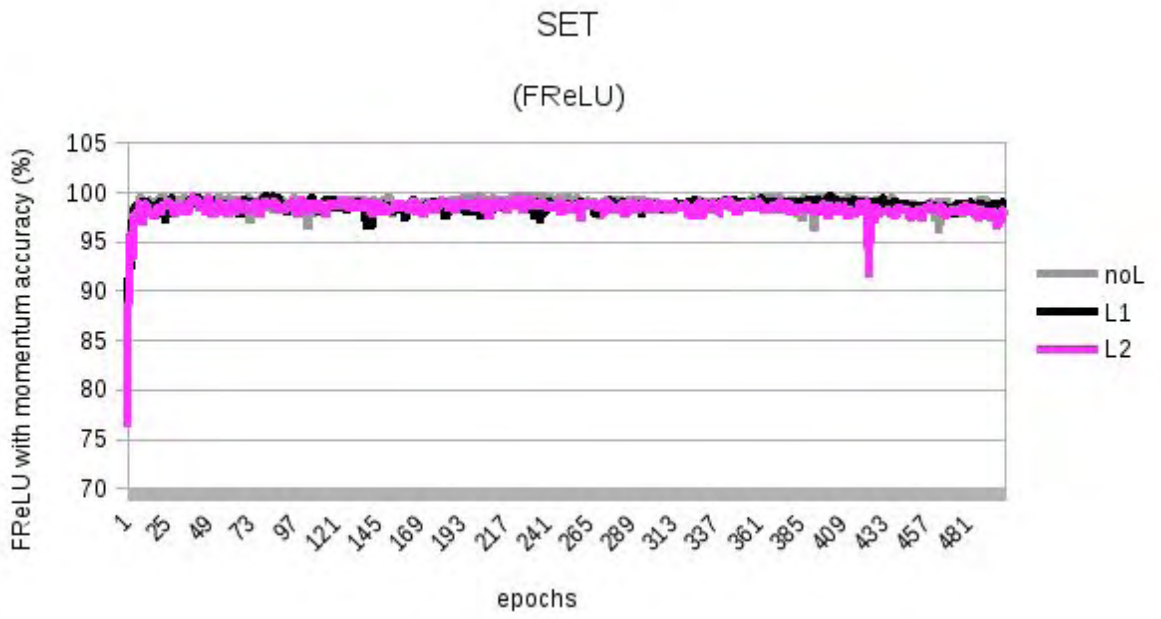


Figure 5.10: SET accuracy, using FReLU activation function and COIL20.mat file.

ReLU	NoL	98.64	48 mins
ReLU	L2	98.92	55 mins
ReLU	L1	99.01	1 h 28 mins
FReLU	NoL	98.8	1 h 26 mins
FReLU	L2	98.28	1 h 31 mins
FReLU	L1	98.55	2 h

Table 5.6: Statistics of SET algorithm using COIL20.mat file.

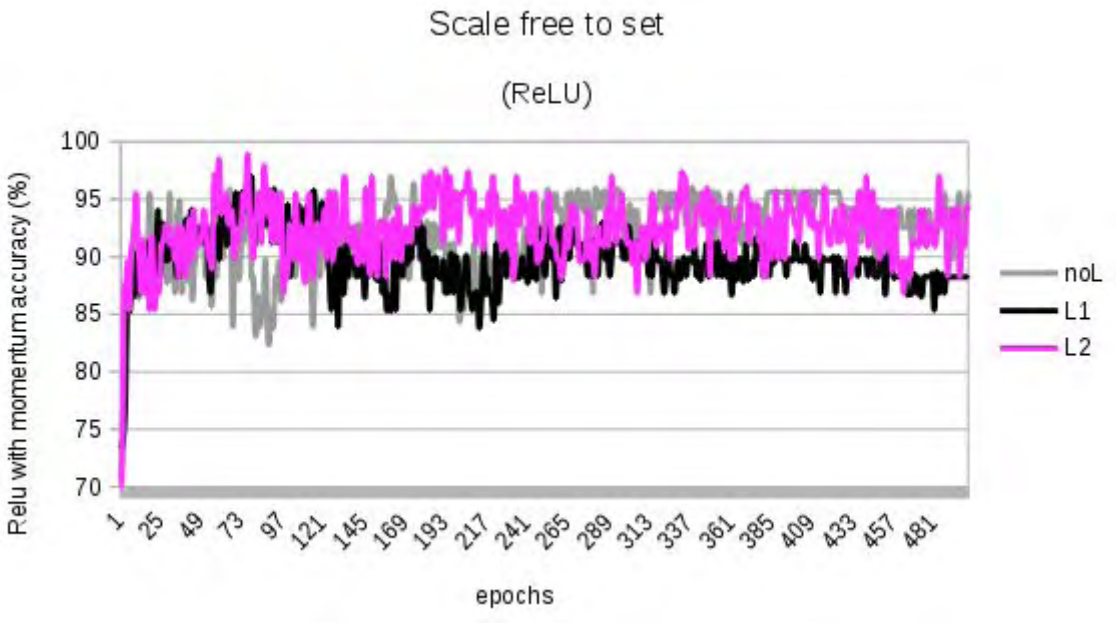


Figure 5.11: Scale Free Set accuracy, using ReLU activation function and lung.mat file.

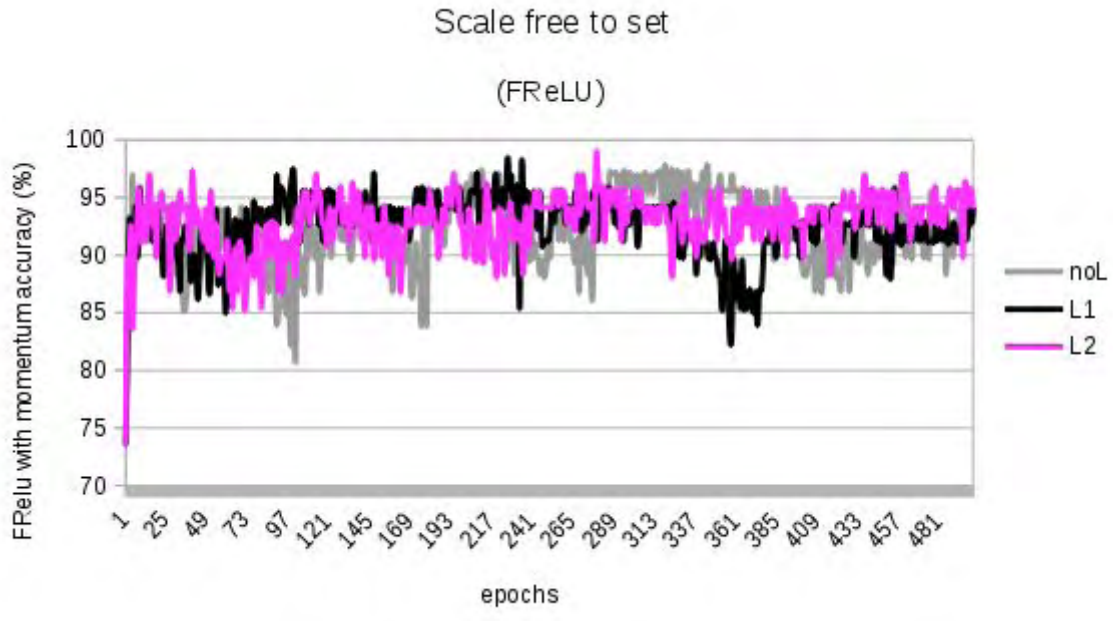


Figure 5.12: Scale Free Set accuracy, using FReLU activation function and lung.mat file.

Figures 5.11 and 5.12 display the accuracy that the Scale Free to SET algorithm achieves, regarding the epochs. In Figure 5.11, we see the results that ReLU activation produces for lung.mat dataset. It seems that the accuracy varies from

70% to 99% and we conclude that the algorithm has a better performance, when L2 regularization is used. This happens because L2 regularization makes the loss function smooth, which means it is easier to find the optimum solution of this function (where the derivative of loss function is equal to zero), which means more successful weight update. So, it seems that L2 manages to decrease the noise in the training data and so as the estimated coefficients (weights) can generalize well to the future data. According to the time, as we can see in Table 5.7, the results are very satisfying. This algorithm achieves high accuracy very quickly. Specifically, using ReLU activation function, the training time, without any regularization technique, is 10 minutes and its accuracy reaches approximately the 92%. Similarly, the according time, using L2 is 1 minute bigger and so as the accuracy, which reaches approximately 93%. Applying L1 parameter, the network achieves approximately 90% accuracy in about 15 minutes. It is obvious that the training time in cases where no regularization is used, is smaller due to the fact that algorithm makes less calculations (loss function doesn't have any penalty factor), regarding L1 or L2. Also, L1 regularization doesn't have as good performance as L2 and NoL do, because L1-regularized loss function is non-smooth. In other words, it's not differentiable at zero and as the optimization theory says, the optimum solution is difficult to find, in this way. In a similar way, applying FReLU transfer function, the algorithm reaches approximately 92% accuracy in about 15 minutes, using NoL and L2 parameters, whereas applying L1 regularization, algorithm's performance is quite the same but the training time is about 5 minutes greater than ReLU, due to computational complexity in FReLU code.
In particular, Table 5.7 shows the exact results, briefly.

ReLU	NoL	91.8	10 mins
ReLU	L2	92.3	11 mins
ReLU	L1	89.6	15 mins
FReLU	NoL	92.1	13 mins
FReLU	L2	92.9	16 mins
FReLU	L1	92.4	21 mins

Table 5.7: Statistics of Scale Free to Set algorithm using lung.mat file.

In conclusion, we can see that algorithm's performance, for this amount of data (for this number of inputs, encoded in the way, mentioned in paragraph 5.1.1) is good. Algorithm succeeds high accuracy, which means, the error between the target (correct output) and the predicted value is decreased, during the epochs. In the sections that follow, this performance is tested, using different datasets.

The experiment is continued, by using a smaller dataset with fewer samples and features, in order to estimate algorithm' s performance.

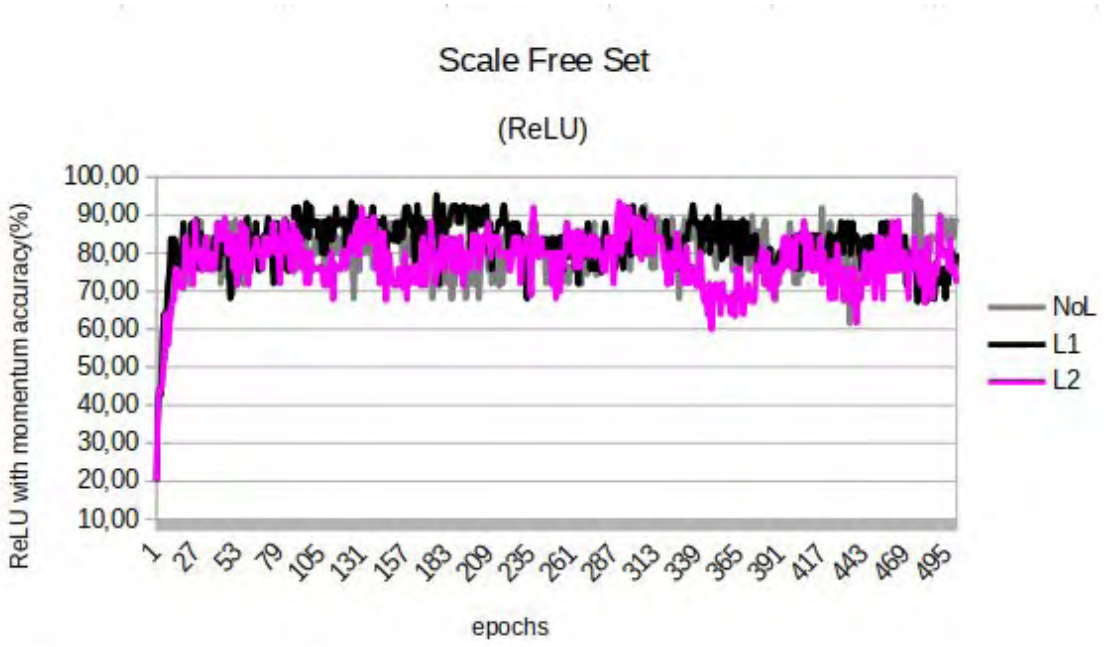


Figure 5.13: Scale Free Set accuracy, using ReLU activation function and lung_discrete.mat file.

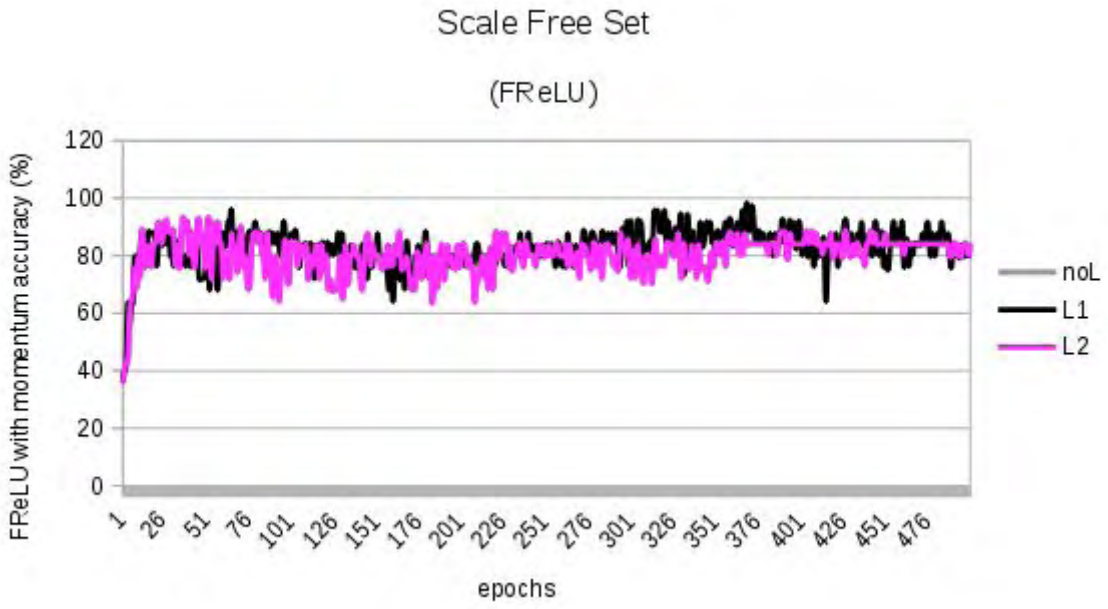


Figure 5.14: Scale Free Set accuracy, using FReLU activation function and lung_discrete.mat file.

In Figures 5.13 and 5.14, we notice that the accuracy, in this dataset, for both ReLU and FReLU activation functions and the parameters, used, is about 80% and the training time, needed is about 6 minutes (as Table 5.8 shows). It is obvious that the algorithm's performance is decreased, which means that neural network isn't able to learn from a small amount of data, correctly. Although the fact that the accuracy is decreased, still, remains a good one, especially, regarding the training time, needed. We conclude that L1 regularization, combined with FReLU activation, achieves better accuracy.

ReLU	NoL	80	6 mins
ReLU	L2	77.6	7 mins
ReLU	L1	82.4	8 mins
FReLU	NoL	80.3	6 mins
FReLU	L2	80.3	6 mins
FReLU	L1	82.5	7 mins

Table 5.8: Statistics of Scale Free to Set algorithm using lung_discrete.mat file.

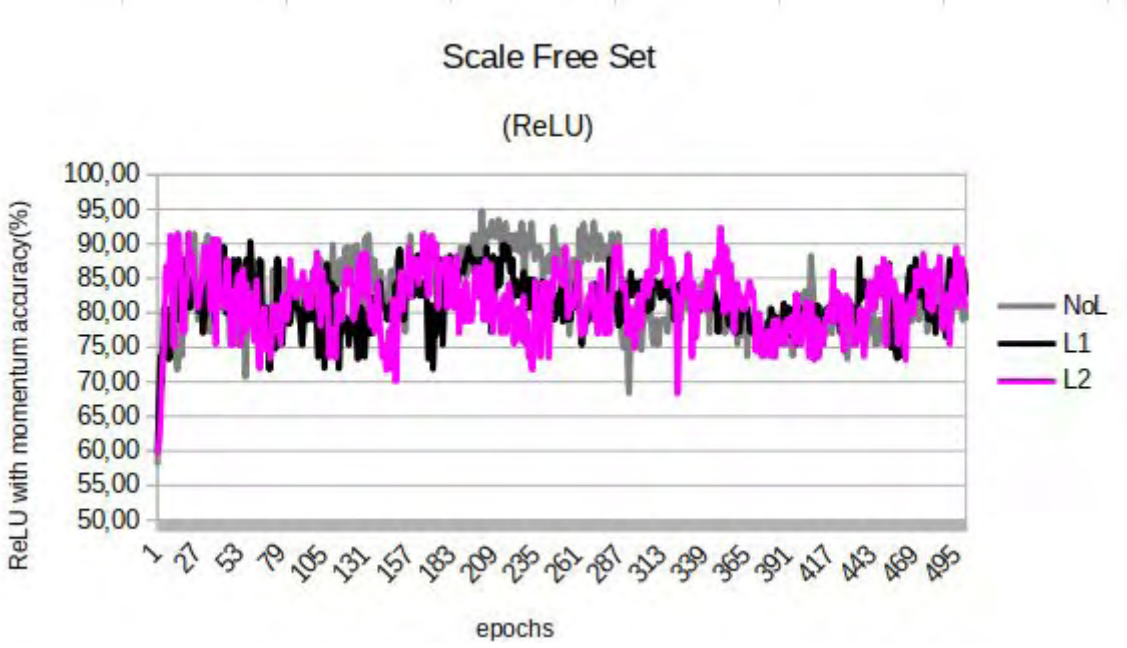


Figure 5.15: Scale Free Set accuracy, using ReLU activation function and TOX_171.mat file.

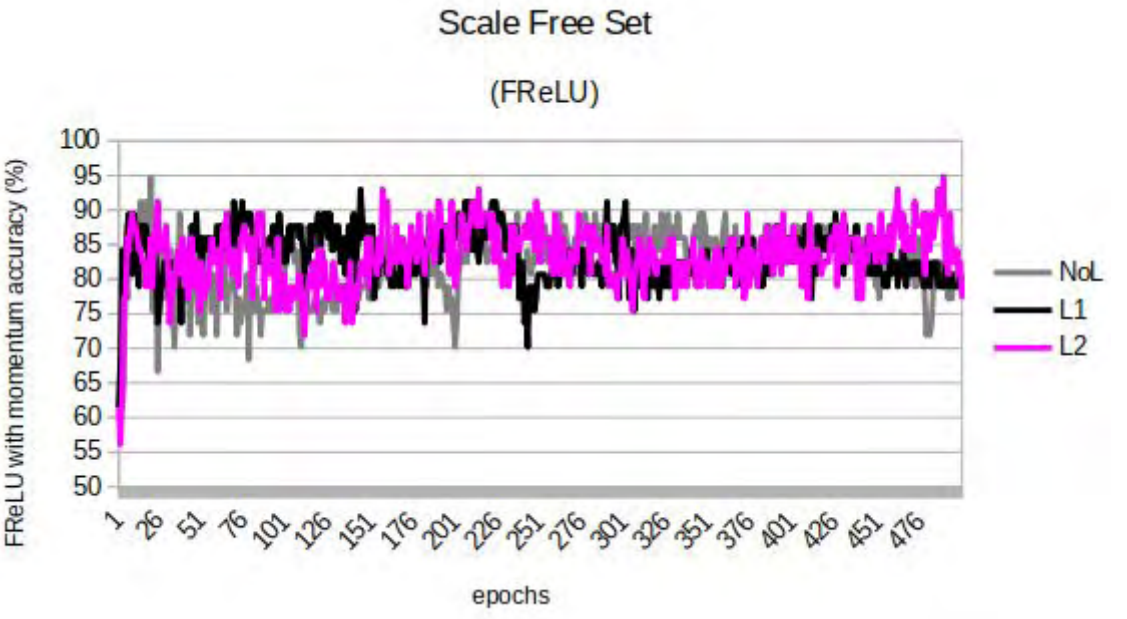


Figure 5.16: Scale Free Set accuracy, using FReLU activation function and TOX_171.mat file.

Figures 5.15 and 5.16, illustrate the accuracy of Scale to SET algorithm, using a bit larger dataset than lung (concerning the features). In all cases, the accuracy is about 82%, as it is shown in Table 5.9. We notice that ReLU with NoL combination has higher accuracy than both L2 and L1, which occurs probably due to the fact that in this type of well-structured network in correlation with ReLU, information is efficiently-distributed. Moreover, ReLU implementation takes less

training time than FReLU, due to its better computational complexity. On the contrary, concerning FReLU, the accuracy, as depicted in Figure 5.16, is higher when L1 or L2 regularization technique is used.

ReLU	NoL	82.5	10 mins
ReLU	L2	81.6	12 mins
ReLU	L1	81.4	18 mins
FReLU	NoL	82.3	16 mins
FReLU	L2	83.3	14 mins
FReLU	L1	83.3	22 mins

Table 5.9: Statistics of Scale Free to Set algorithm, using TOX_171.mat file.

In this case, we test our algorithm with the biggest dataset we have (concerning the features).

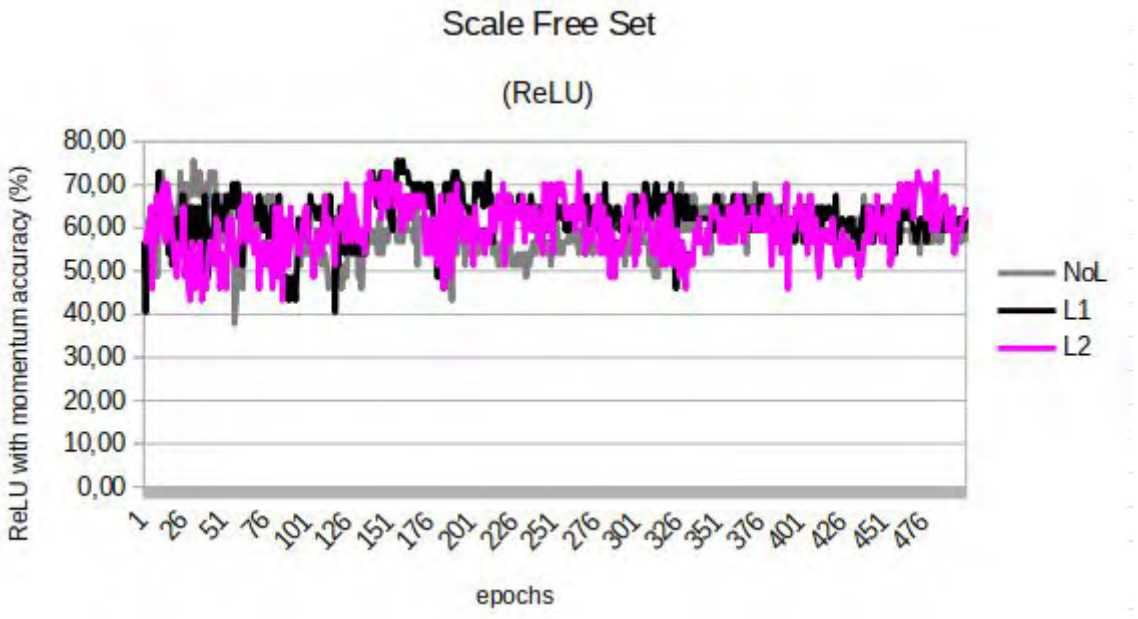


Figure 5.17: Scale Free Set accuracy, using ReLU activation function and CLL_SUB_111.mat file

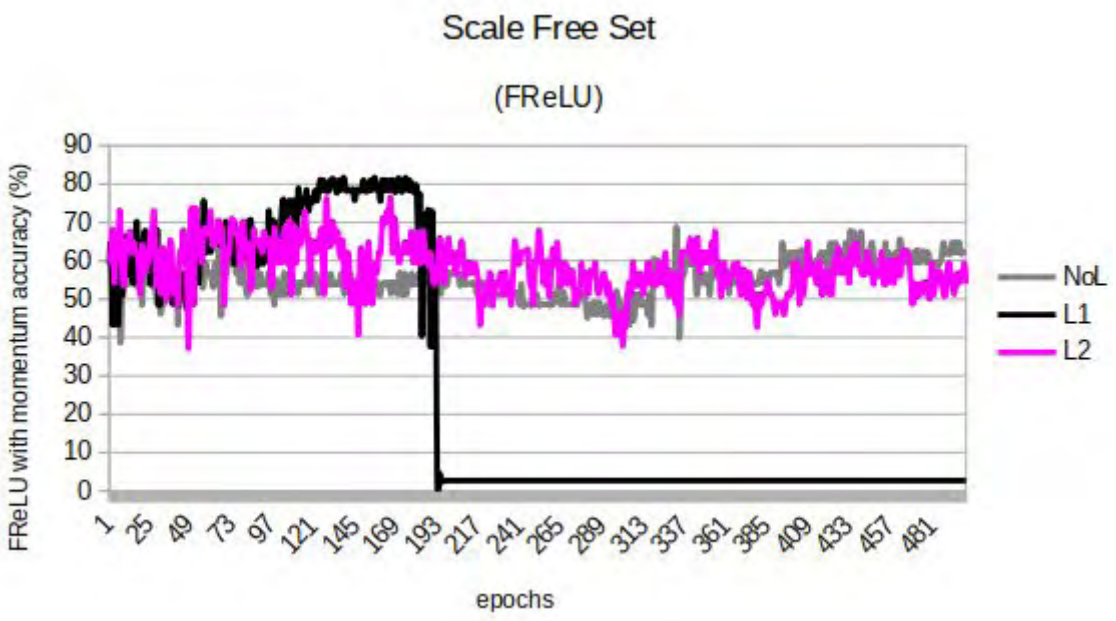


Figure 5.18: Scale Free Set accuracy, using FReLU activation function and CLL_SUB_111.mat file.

Although the dataset has a great amount of features, it classifies its data, only, into three classes, which means, it takes less time to correlate the predicted value to one of the classes. According to the accuracy, the results are not satisfying, because of the fact that the big number of features (much more than the neurons in every hidden layer) makes a more complex-structured network and the information, saved in every node may not be efficiently distributed to the system. However, in case of L1 in correlation with FReLU, the results are disappointing. In particular, as depicted in Figure 5.18, the L1 regularized curve seems to falls off, abruptly because of a code warning (appears NaN values). Probably, the cause of this problem is the large amount of data, given for processing.

Table 5.10 shows detailed information about Scale Free to SET performance, using CLL_SUB_111.mat file.

ReLU	NoL	58.9	15 mins
ReLU	L2	60	14 mins
ReLU	L1	62.4	19 mins
FReLU	NoL	55.4	18 mins

FReLU	L2	58.2	15 mins
FReLU	L1	27.6	17 mins

Table 5.10: Statistics of Scale Free to SET algorithm, using CLL_SUB_111.mat file.

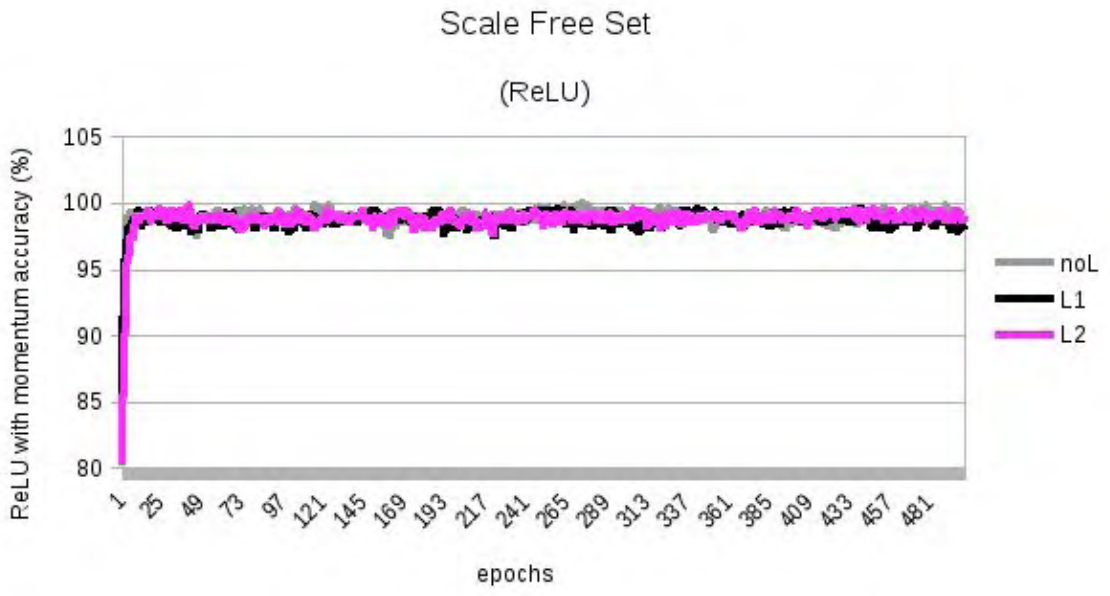


Figure 5.19: Scale Free Set accuracy, using ReLU activation function and COIL20.mat file.

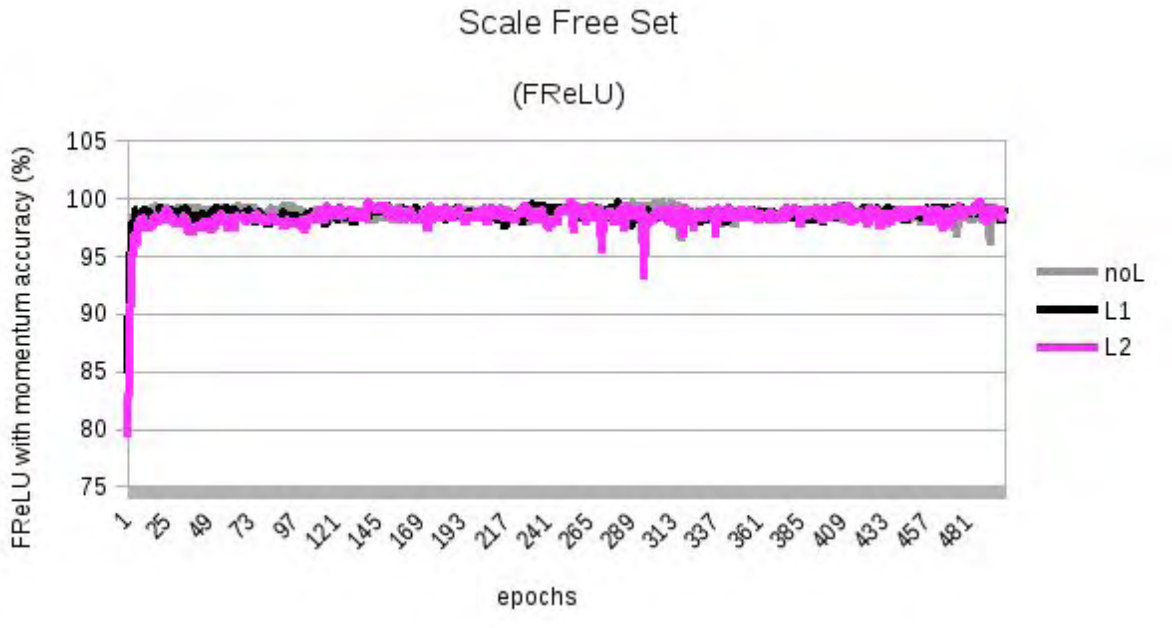


Figure 5.20: Scale Free Set accuracy, using FReLU activation function and COIL20.mat file.

Figures 5.19 and 5.20 depict the accuracy of the algorithm using a different kind of dataset. In this case we use image data to see how the neural network can respond to such predictive problems. According to the plots, we can see that algorithm has stable behaviour which is due to the fact that this dataset has higher amount of samples than biological datasets and so the neural network is trained for a large number of inputs and learns to recognize many images. Thus, algorithm achieves very high accuracy, reaching even 100% in some epochs.
When combination ReLU activation function and NoL or L2 regularization is used, algorithm takes less training time (approximately 40 mins). In the other cases, neural network needs much more time to be trained, about 1-2 hours, as we can observe in Table 5.11. Compared to the other datasets, here, the algorithm takes more training time because COIL20.mat file has greater amount of classes (20 instead of 7 in lung_discrete.mat file).

ReLU	NoL	98.91	37 mins
ReLU	L2	98.79	45 mins
ReLU	L1	98.68	1 h 9 mins
FReLU	NoL	98.6	1 h 15 mins
FReLU	L2	98.38	1 h 29 mins
FReLU	L1	98.57	1 h 43 mins

Table 5.11: Statistics of Scale Free to SET algorithm, using COIL20.mat file.

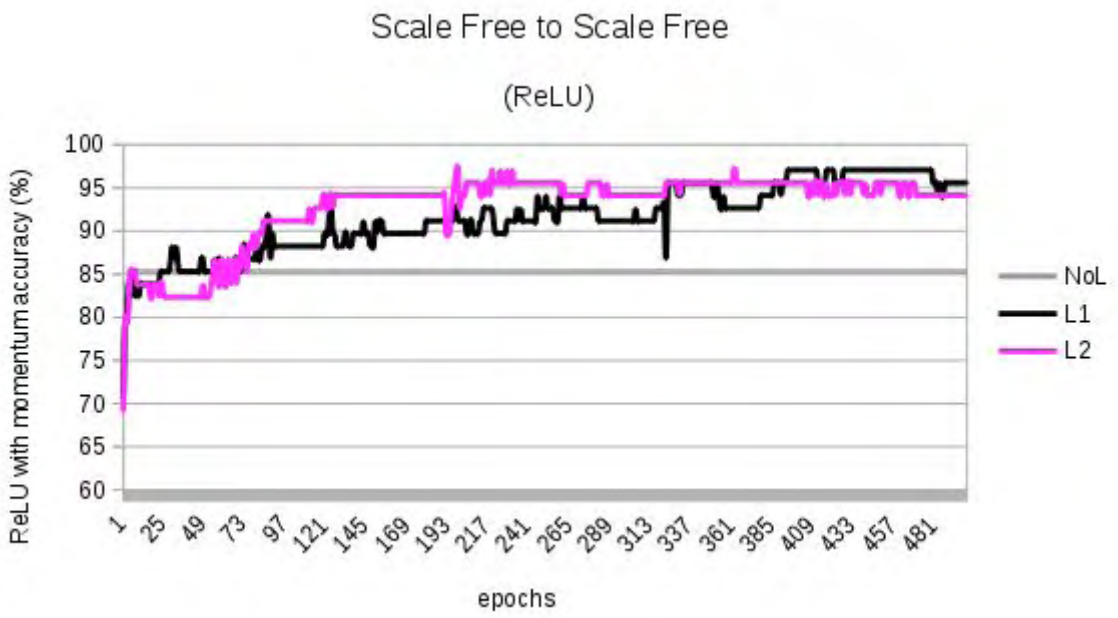


Figure 5.21: Scale Free to Scale Free accuracy, using ReLU activation function and lung.mat file.

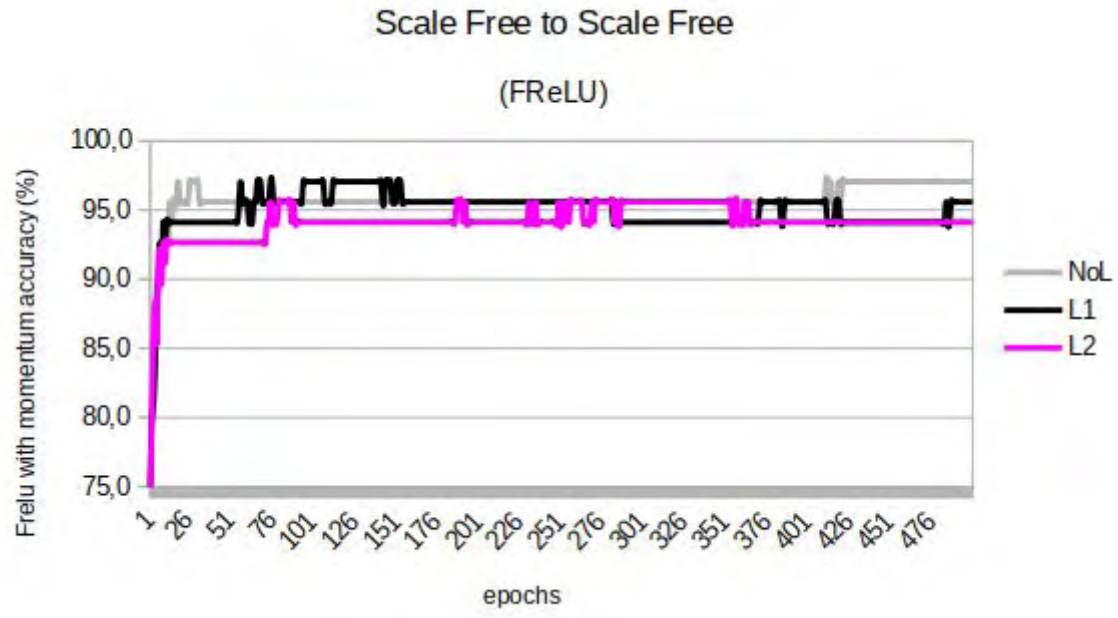


Figure 5.22: Scale Free to Scale Free accuracy, using FReLU activation function and lung.mat file.

In this algorithm, we train the network by starting form an exact scale-free network and through the procedure of training, using back propagation, we end up constructing a same one. This means that network is strictly constructed

following power law and hence the randomness is decreased, which means better communication between the nodes is achieved. According to the charts, this variant of the algorithm, has 95,6% accuracy, using FReLU transfer function and without any regularization. So, taking into consideration that FReLU has faster convergence than ReLU [4], concerning, also, the network topology, we conclude that the less random a network structure is, the higher performance the network gets, without any regularization technique. In addition, algorithm takes 29 minutes (with 92% accuracy) to train the network, using ReLU (this function is computational efficient by just outputting zero for negative inputs [4]) and 35 minutes while implementing the FReLU function, which is still better in both time and accuracy than the SET code. The Table 5.12 shows analytically the results, concerning different parameters in the algorithm.

ReLU	NoL	85.2	29 mins
ReLU	L2	92.8	30 mins
ReLU	L1	91.5	32 mins
FReLU	NoL	95.6	35 mins
FReLU	L2	94.1	39 mins
FReLU	L1	94.9	41 mins

Table 5.12: Statistics of Scale Free to Scale Free algorithm, using lung.mat file.

Figure 5.23: Scale Free to Scale Free accuracy, using ReLU activation function and lung_discrete.mat file.

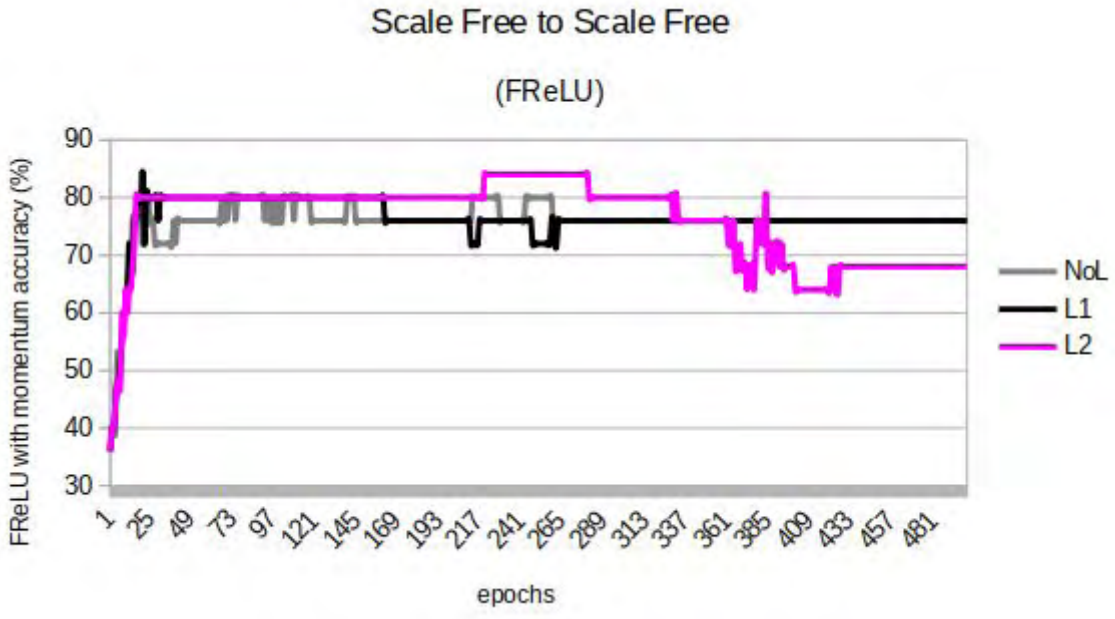


Figure 5.24: Scale Free to Scale Free accuracy, using FReLU activation function and lung_discrete.mat file.

Comparing to lung.mat statistics, the results of this file is not enough encouraging. We have lower performance in, approximately, similar time. Although the number of features is smaller, the time, needed, isn't as less enough as we expected to be. This happens owing to the fact that there are more classes, which algorithm has to compare the predicted value with. According to the accuracy, ReLU

implementation has better results, as Figure 5.23 illustrates. It's worth observing that in ReLU graph, the accuracy start from very small values (about 20%) and manages to reach accuracy of up to 80%, after the training procedure. In FReLU, accuracy, also, increases according to the epochs, but in this case the accuracy starts from about 35% (a bit higher than ReLU).

ReLU	NoL	80	15 mins
ReLU	L2	81.4	15 mins
ReLU	L1	81.4	18 mins
FReLU	NoL	75.9	18 mins
FReLU	L2	76	18 mins
FReLU	L1	76.3	20 mins

Table 5.13: Statistics of Scale Free to Scale Free algorithm, using lung_discrete.mat file.

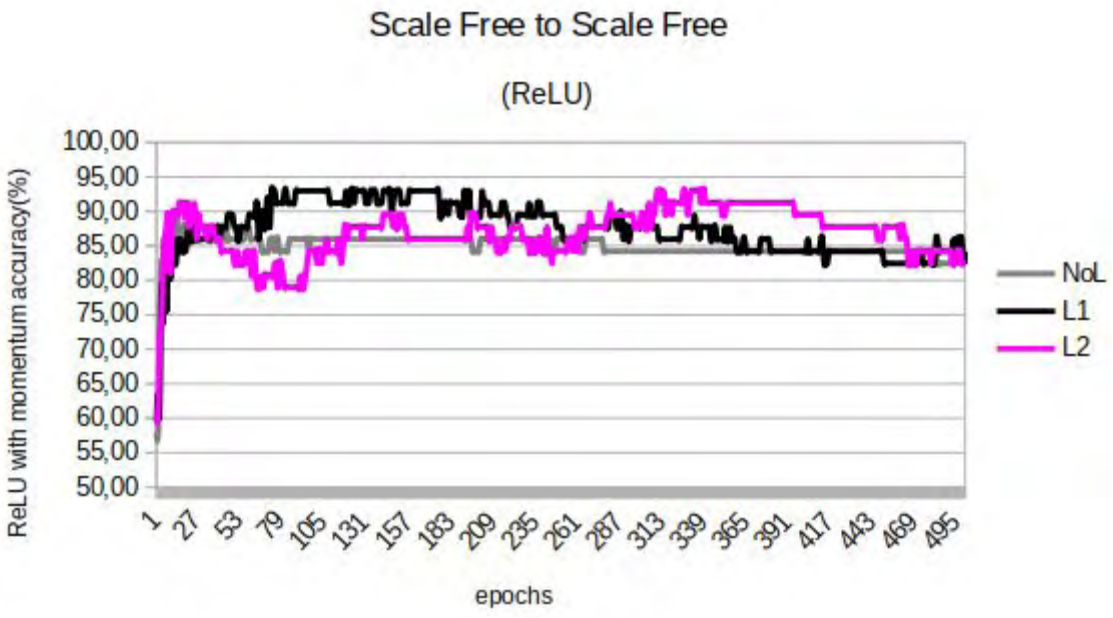


Figure 5.25: Scale Free to Scale Free accuracy, using ReLU activation function and TOX_171.mat file.

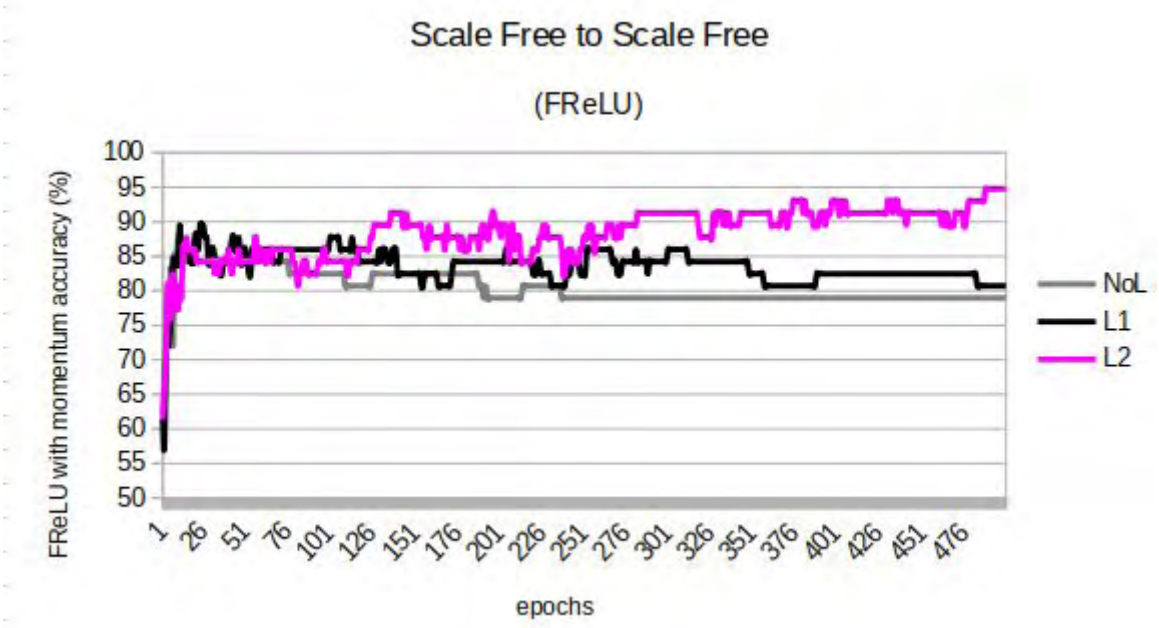


Figure 5.26: Scale Free to Scale Free accuracy, using FReLU activation function and TOX_171.mat file.

In this case, we also, notice that the values regarding accuracy, are small at the beginning and get higher after the first 25 epochs. This means that algorithm fits the TOX_171.mat file data efficiently, during the training process (we see that L2 combined with FReLU reaches 95% accuracy in the last epoch). Due to the large amount of features (5748 features), algorithm takes more time to train the network, than in cases of lung.mat (3312 features) and lung_discrete.mat (325 features) files. Specifically, algorithm reaches approximately 90% accuracy, in about 40 minutes (ReLU-L1 and FReLU-L2), which means that this variant of our concept, can be efficient enough in respect to bigger datasets.

ReLU	NoL	84.8	34 mins
ReLU	L2	86.8	37 mins
ReLU	L1	87.5	39 mins
FReLU	NoL	80.4	38 mins
FReLU	L2	88.3	42 mins
FReLU	L1	83.5	1 h 19 mins

Table 5.14: Statistics of Scale Free to Scale Free algorithm, using TOX_171.mat file.

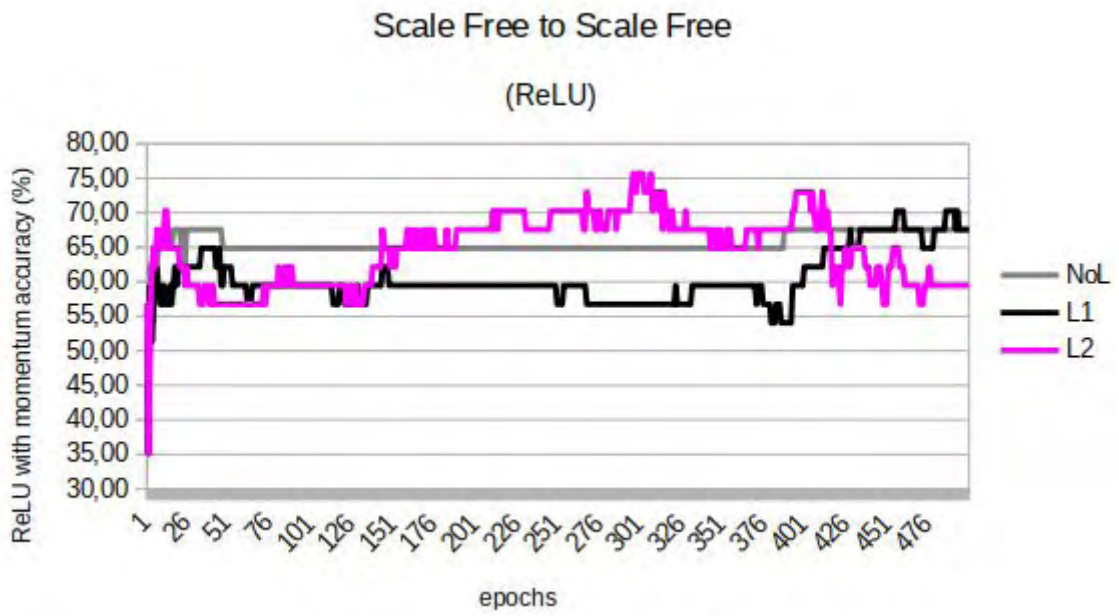


Figure 5.27: Scale Free to Scale Free accuracy, using ReLU activation function and CLL_SUB_111.mat file.

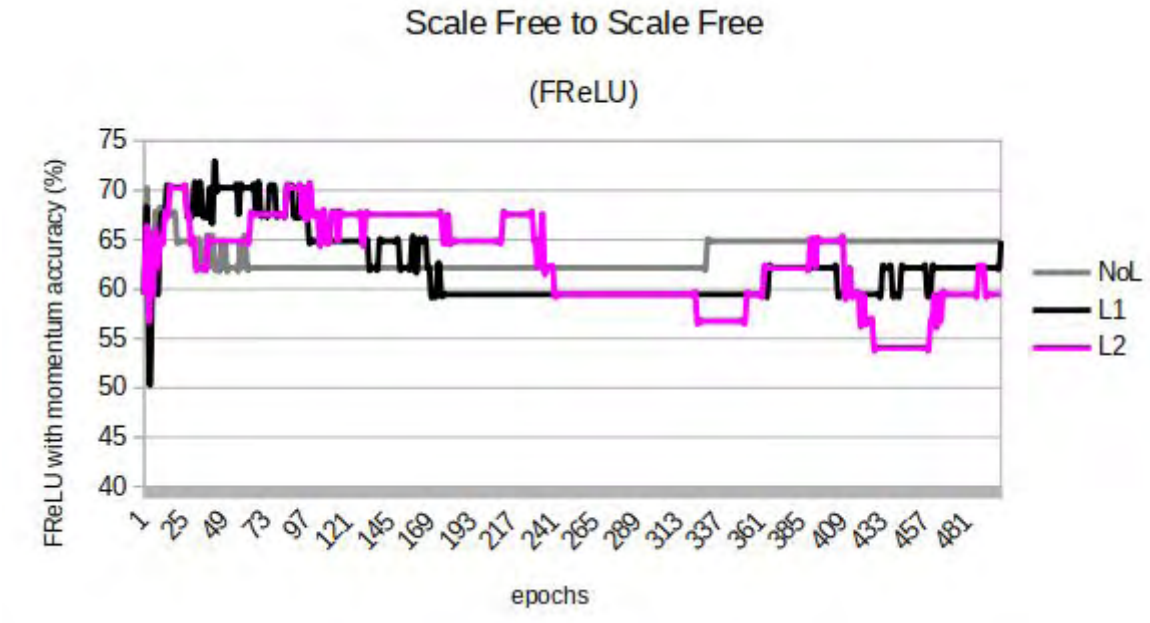


Figure 5.28: Scale Free to Scale Free accuracy, using FReLU activation function and CLL_SUB_111.mat file.

Taking into consideration the results of Scale Free to SET algorithm, using CLL_SUB_111.mat file, we see that in this algorithm, this file responds better to the

network, with respect to the accuracy in every epoch. However, the training time is much larger, owing to the fact that algorithm has to process a large number of data. At the same time, in this variant, we construct, during the training part, an absolute scale-free network, which means it takes more time to process the corresponding data than Scale Free to SET algorithm, in which SET is a type of scale-free, not a strictly-constructed one. According to the statistics, the algorithm tends to generalize well from its training data to unseen data when no regularization method is used, as both Figures 5.27, 5.28 and Table 5.15 show.

ReLU	NoL	65.5	1 h
ReLU	L2	64.5	1 h 5 mins
ReLU	L1	60.4	1 h 7 mins
FReLU	NoL	63.3	1 h
FReLU	L2	62.6	1 h 12 mins
FReLU	L1	62.4	1 h 5 mins

Table 5.15: Statistics of Scale Free to Scale Free algorithm, using CLL_SUB_111.mat file.

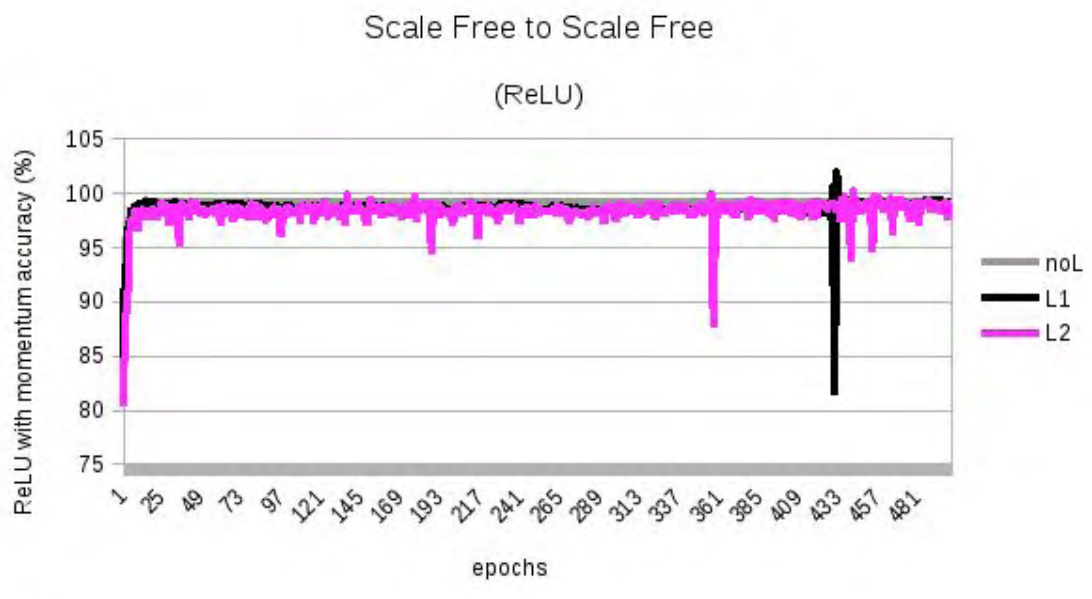


Figure 5.29: Scale Free to Scale Free accuracy, using ReLU activation function and

COIL20.mat file.

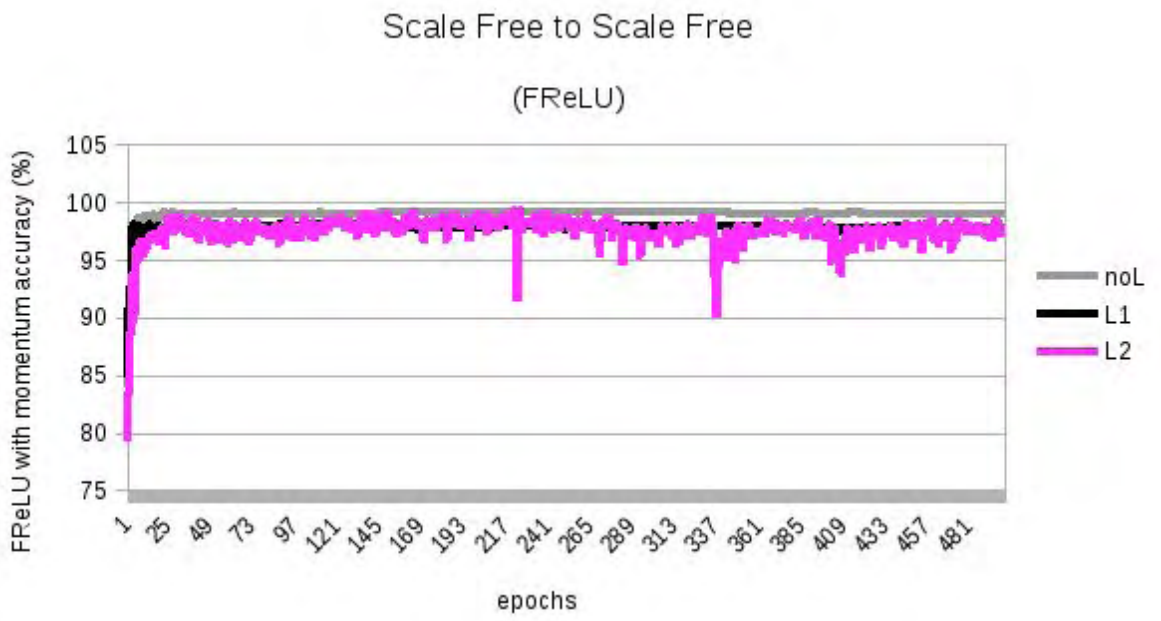


Figure 5.30: Scale Free to Scale Free accuracy, using FReLU activation function and COIL20.mat file.

In this case, we test Scale Free to Scale Free algorithm on image data. We notice that it can respond very well and achieves high values of accuracy (about 98%). Specifically, using ReLU activation function and no regularization, algorithm has 99% accuracy in 53 minutes. The lower accuracy, but still very satisfying, is 97.48% in 1 hour and 51 minutes, when FReLU and L2 regularization combination is used. We conclude that COIL20 dataset helps neural network, in this algorithm too, to get very close to the desired output (target value), due to the large amount of samples.

ReLU	NoL	99.09	53 mins
ReLU	L2	98.26	1 h 19 mins
ReLU	L1	98.61	1 h 22 mins
FReLU	NoL	98.95	1 h 29 mins
FReLU	L2	97.48	1 h 51 mins

FReLU	L1	97.74	1 h 58 mins

Table 5.16: Statistics of Scale Free to Scale Free algorithm, using COIL20.mat file.

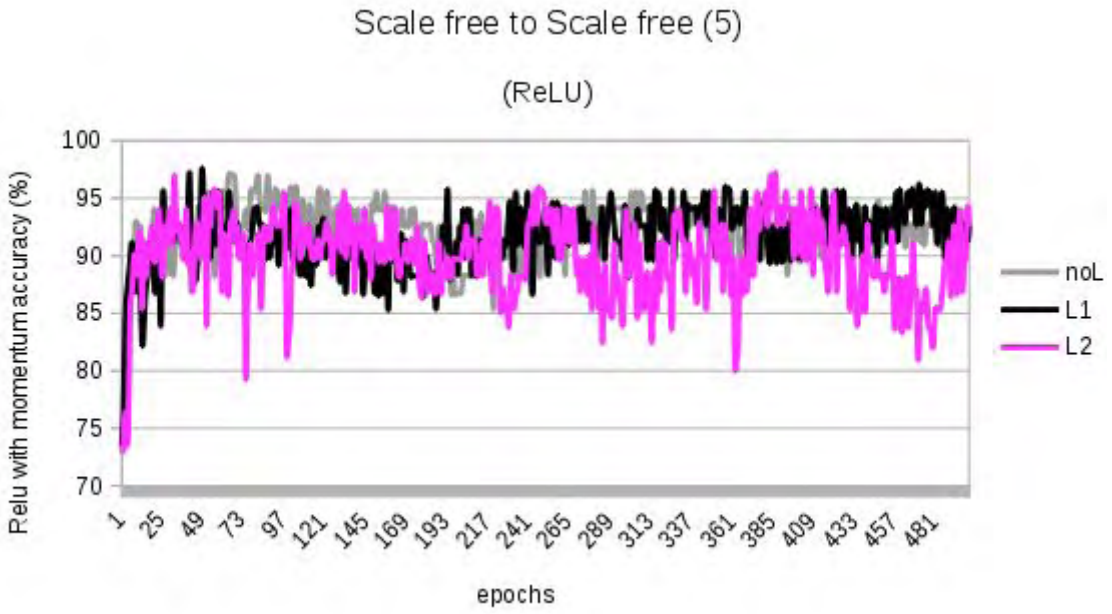


Figure 5.31: Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and lung.mat file.

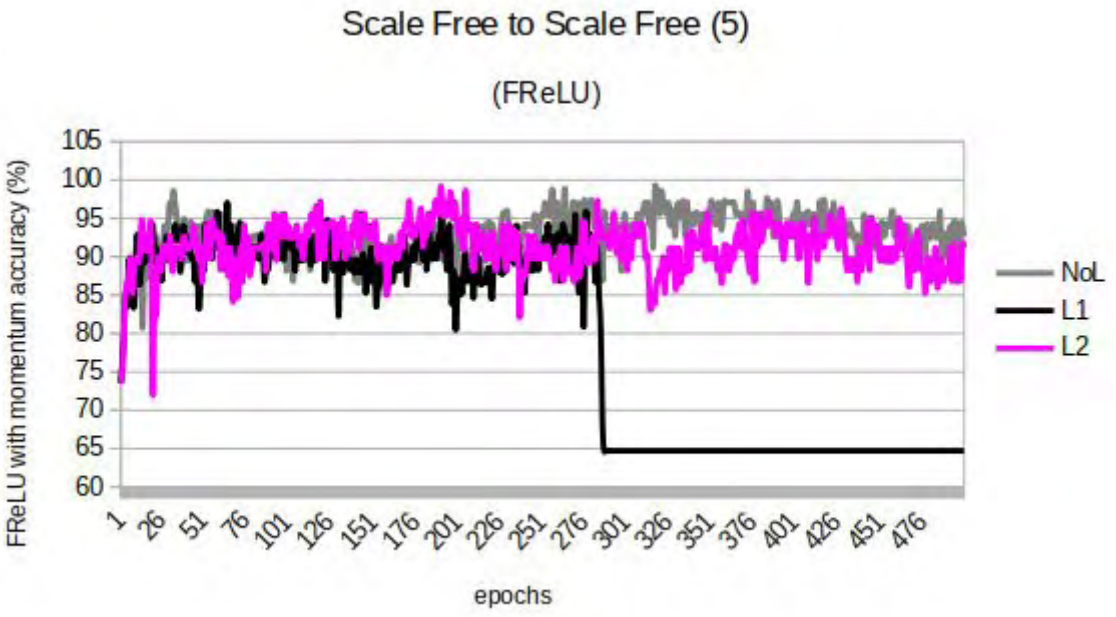


Figure 5.32: Scale Free to Scale Free (5 strongest nodes) accuracy, using FReLU

activation function and lung.mat file.

Figures 5.31 and 5.32 show the accuracy of a variant of Scale Free to Scale Free algorithm, described in section 1.3. Algorithm achieves approximately 93% accuracy, while implementing FReLU activation function and no regularization techniques, due to the faster convergence of this function. Furthermore, both ReLU and FReLU activations affect positively the network performance, except the case where L1 regularization, combined with FReLU is used. We have approximately 79% accuracy. We conclude that L1 regularization technique makes the graph curve falls off abruptly, because of a code warning in weight update (appears NaN values). Concerning the time, algorithm takes approximately the same time to train the network as Scale-Free to Scale-Free version does.

ReLU	NoL	92	28 mins
ReLU	L2	89.8	30 mins
ReLU	L1	90.6	37 mins
FReLU	NoL	92.8	35 mins
FReLU	L2	91.2	37 mins
FReLU	L1	79	37 mins

Table 5.17: Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using lung.mat file.

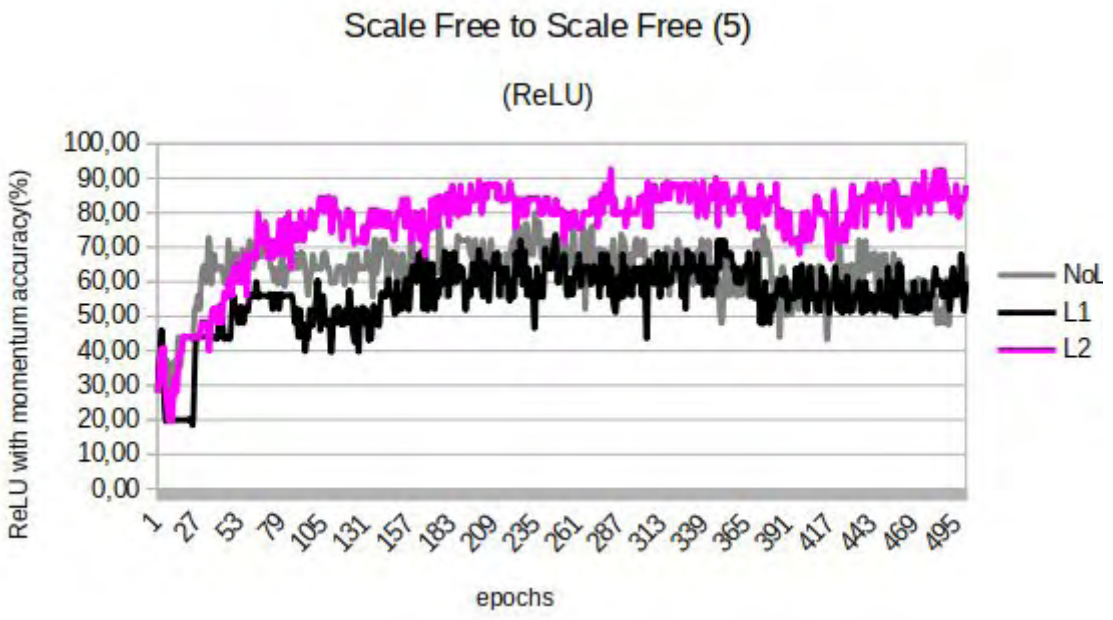


Figure 5.33: Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and lung_discrete.mat file.

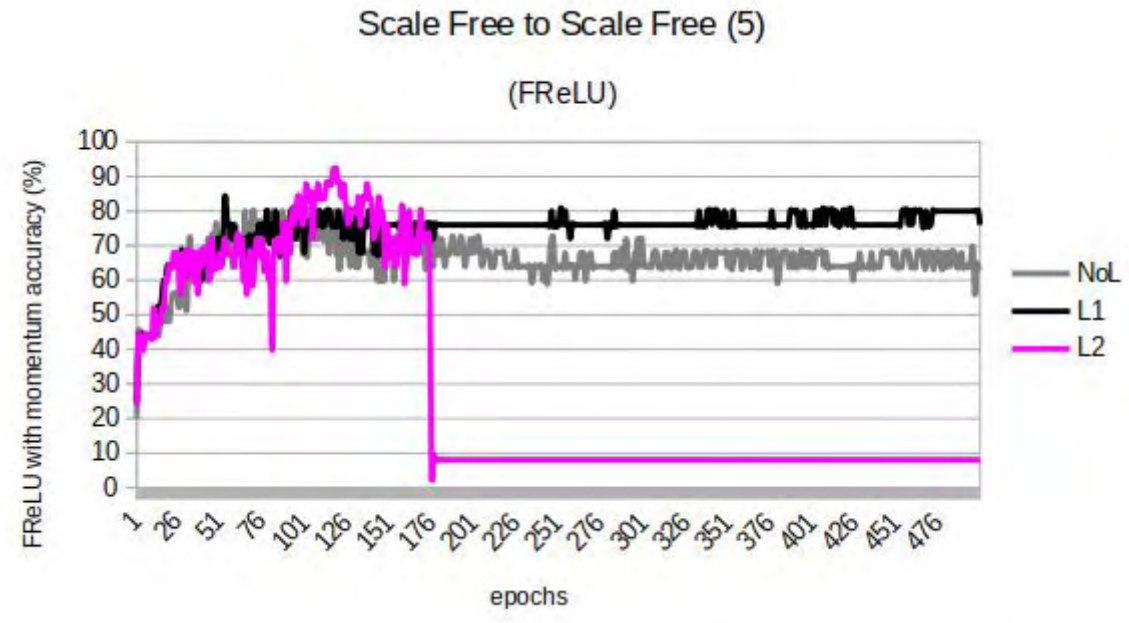


Figure 5.34: Scale Free to Scale Free (5 strongest nodes) accuracy, using FReLU activation function and lung_discrete.mat file.

In this dataset, although the number of classes are bigger than lung file (we have 7 classes), the training procedure is faster due to the fact that we, also have a small amount of input vectors. According to these inputs, using ReLU the algorithm has a better performance than FReLU. Specifically, the combination of ReLu activation and L2 regularization reaches up to 90% accuracy, as shown in Figure 5.33. On

the contrary, FReLU-L2 implementation has a negative impact on the total network system, while we can observe that after the 170 epoch, the accuracy falls off abruptly and stabilizes in about 10% (because of a code warning - this combination seems to appear NaN values). In general, this variant of our concept, for this input dataset, achieves approximately 77% accuracy in about 14 minutes in better case (ReLU-L2), which makes it a moderate choice for modeling these data.

ReLU	NoL	62.4	13 mins
ReLU	L2	76.7	14 mins
ReLU	L1	55.7	13 mins
FReLU	NoL	66.1	13 mins
FReLU	L2	29.5	11 mins
FReLU	L1	74.3	14 mins

Table 5.18: Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using lung_discrete.mat file.

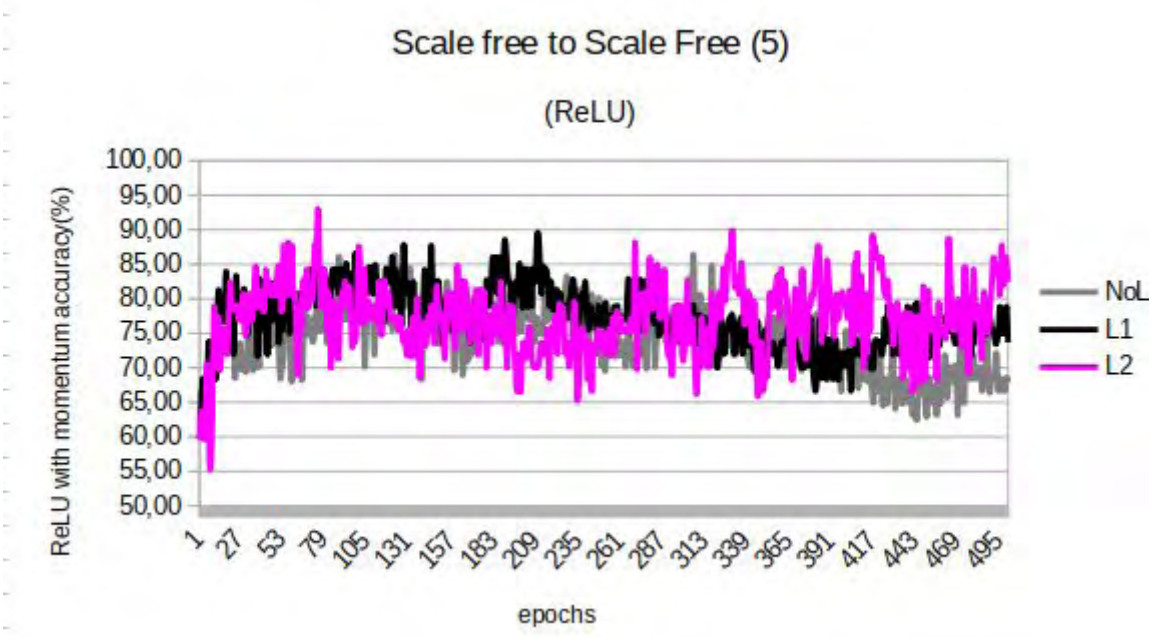


Figure 5.35: Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and TOX_171.mat file.

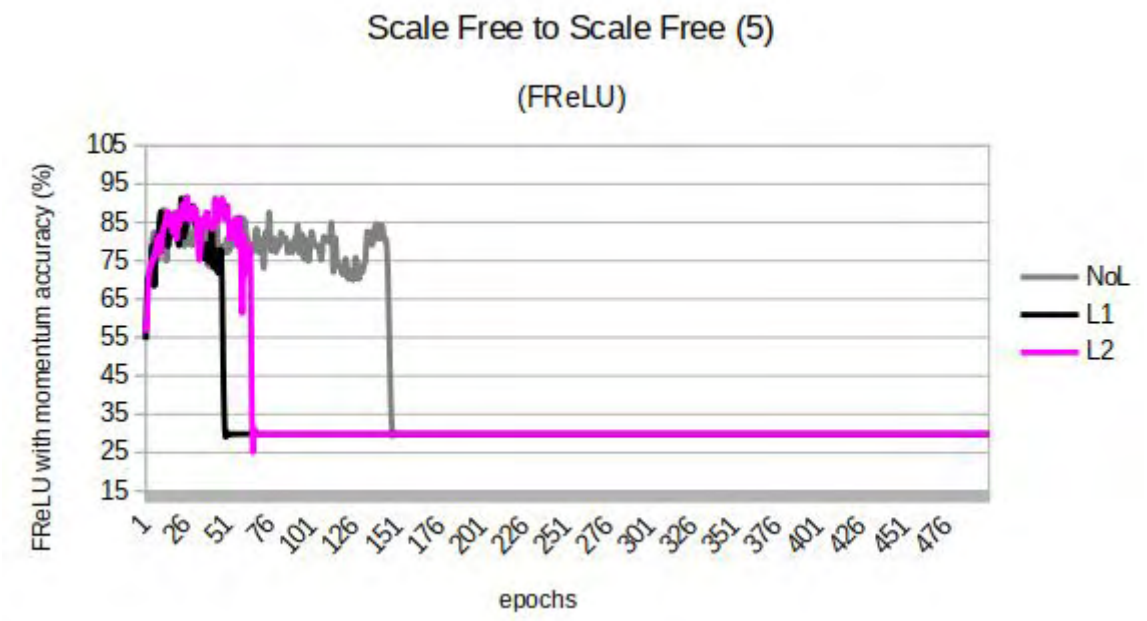


Figure 5.36: Scale Free to Scale Free (5 strongest nodes) accuracy, using FReLU activation function and TOX_171.mat file.

We continue testing Scale Free to Scale Free (5) algorithm, now with a dataset, composed of many instances and features. Observing the Figures 5.35 and 5.36, we can see that when ReLU transfer function is used, there aren't intense fluctuations, concerning accuracy, whereas in cases of implementing FReLU, we have disappointing results, while passing through the 50 epoch with L1-L2 regularization and through the 150 epoch without regularization (while dataset doesn't react well with this type of activation function-causes code warning). According to the time, we see that in cases where the code warning appears, the training phase is completed faster due to the less computations that algorithm has to make (appears no weight values for update processing). Concerning the accuracy, algorithm achieves better performance when a synthesis of ReLu with any kind of regularization is used (accuracy up to 77% in about 50 minutes). Thus, we conclude that the dataset characteristics play an important role in neural network learning process and therefore in network's performance.

ReLU	NoL	74	40 mins
ReLU	L2	77.2	43 mins
ReLU	L1	77.2	50 mins

FReLU	NoL	44	39 mins
FReLU	L2	36.4	33 mins
FReLU	L1	34.4	34 mins

Table 5.19: Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using TOX_171.mat file.

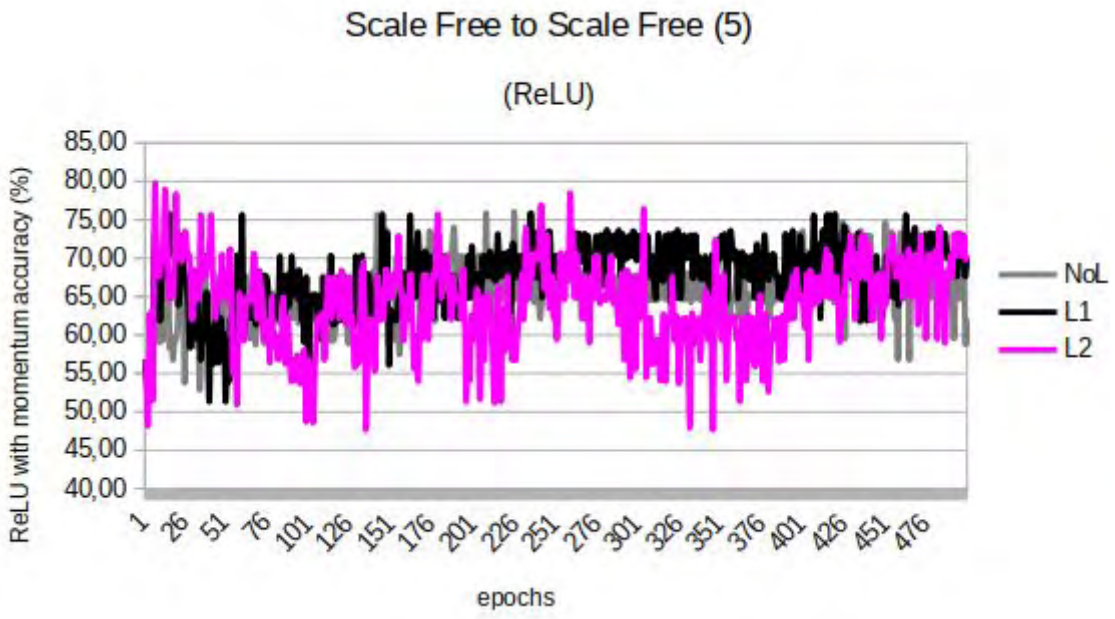


Figure 5.37: Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and CLL_SUB_111.mat file.

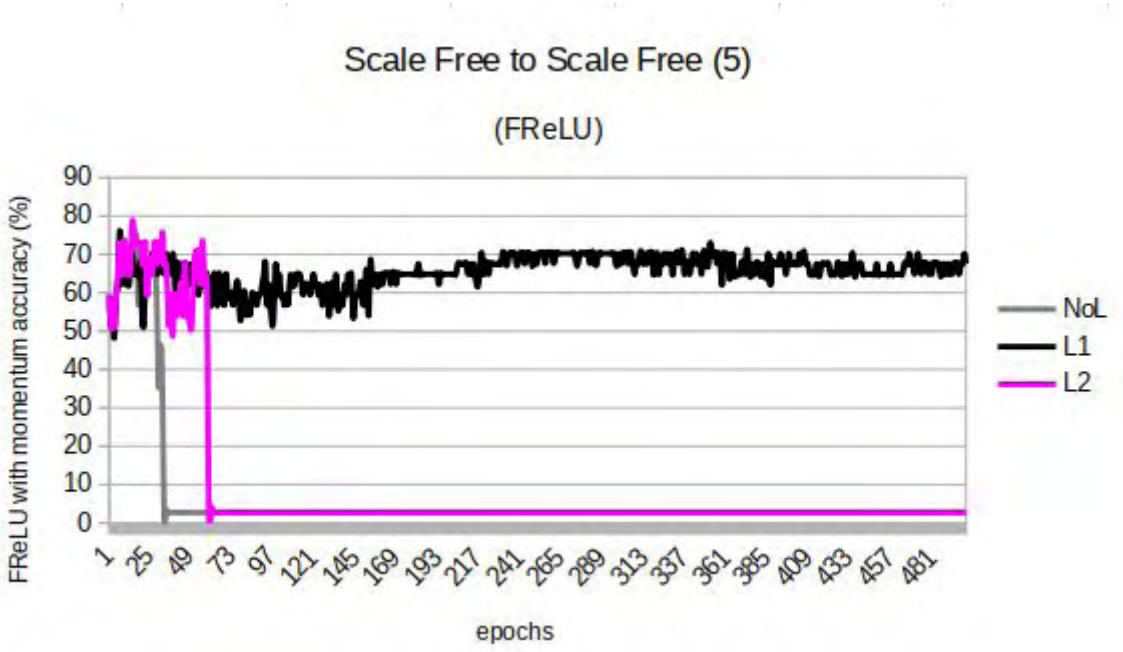


Figure 5.38: Scale Free to Scale Free (5 strongest nodes) accuracy, using FReLU activation function and CLL_SUB_111.mat file.

Regarding the Figures 5.37 and 5.38, we observe that the results for CLL_SUB_111 file has adequate accuracy (about 66%), in some cases (ReLU). This model of network cannot generalize the data with so many features (11320) to the fullest. Furthermore, the time, needed, is reasonable in respect to the calculations the algorithm does (about one hour). Similar accuracy and expected training time are observed in Figure 5.38, in FReLU_L1 combination. Completely disappointing is the fact that not only the FReLU-L2 but also the FReLU-NoL parameters leads the network to have very low success rate, which means it isn't able to make correct enough predictions (about 10% accuracy). As we can see in Table 5.20, the network is trained faster in cases we have inefficient results, perhaps due to the fact that the specific combination of parameters works inhibitly for the network.

ReLU	NoL	66	1 h 15 mins
ReLU	L2	64	1 h 13 mins
ReLU	L1	67.4	1 h 20 mins
FReLU	NoL	6.4	55 mins
FReLU	L2	9.8	57 mins

FReLU	L1	66	1 h 26 mins

Table 5.20: Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using CLL_SUB_111.mat file.

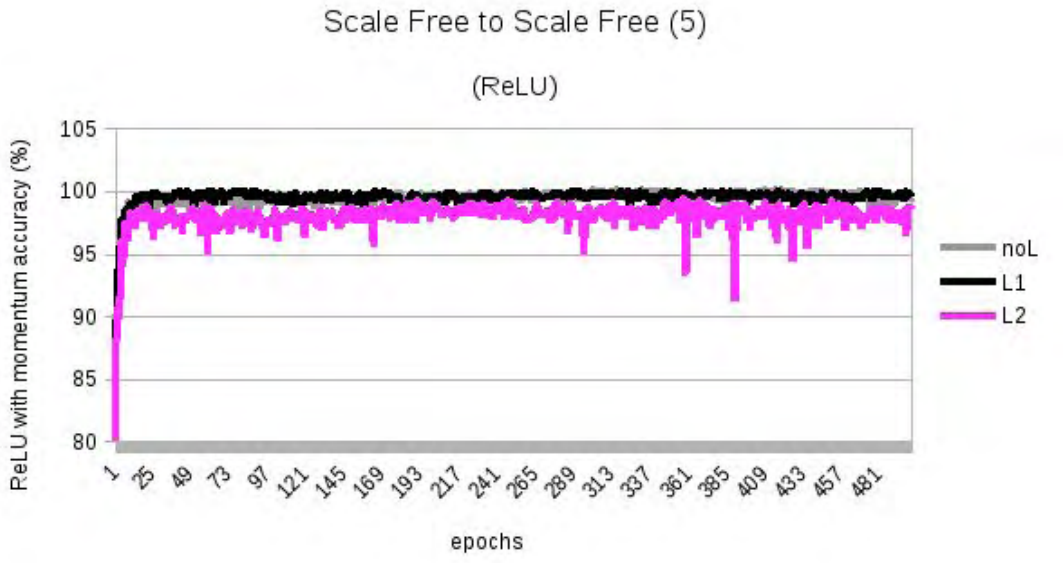


Figure 5.39: Scale Free to Scale Free (5 strongest nodes) accuracy, using ReLU activation function and COIL20.mat file.

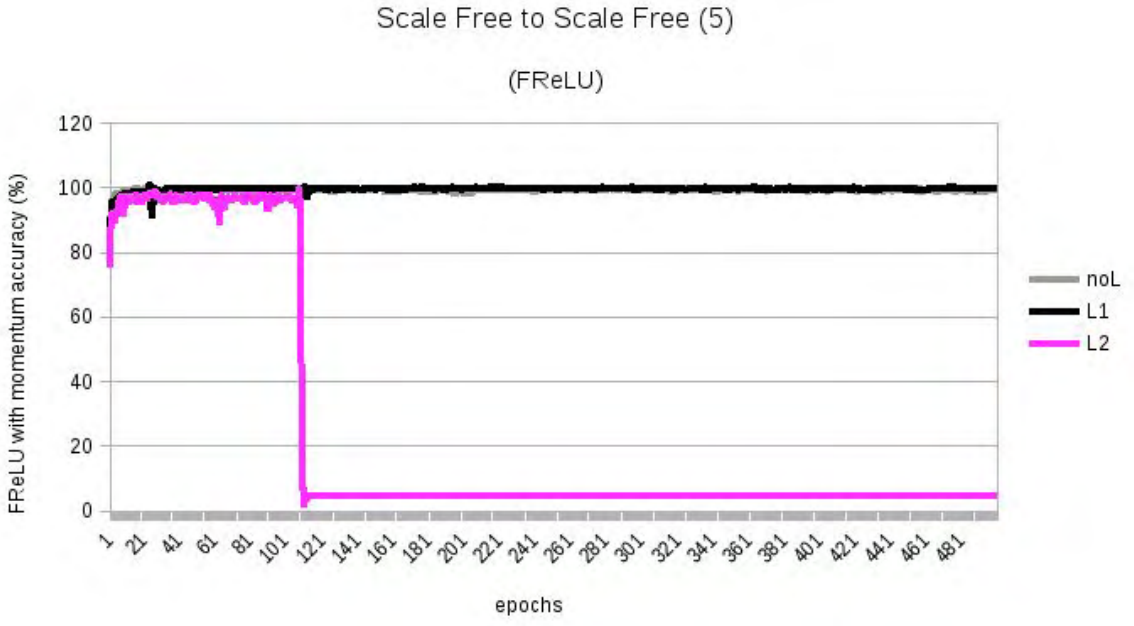


Figure 5.40: Scale Free to Scale Free (5 strongest nodes) accuracy, using FReLU activation function and COIL20.mat file.

Observing Figures 5.39 and 5.40, we notice that algorithm has, in general, stable behaviour in accuracy which varies from 97% to 100%. In Table 5.21, we can see that better accuracy is achieved when ReLU and L1 regularization are used (99.49% in 46 minutes). In the graph in Figure 5.40, there is a fluctuation in L2 curve, which is due to a code warning in weight update (appears NaN values). This warning decreases the average accuracy (value of 24.22%). According to the training time, neural network needs more time, than biological input data, to learn recognize the large amount of input images in this dataset (approximately 45 mins in ReLU and 1 h 15 mins in FReLU).

ReLU	NoL	99.42	42 mins
ReLU	L2	97.98	46 mins
ReLU	L1	99.49	46 mins
FReLU	NoL	99.03	1 h 14 mins
FReLU	L2	24.22	1 h 7 mins
FReLU	L1	99.35	1 h 26 mins

Table 5.21: Statistics of Scale Free to Scale Free (5 strongest nodes) algorithm, using COIL20.mat file.

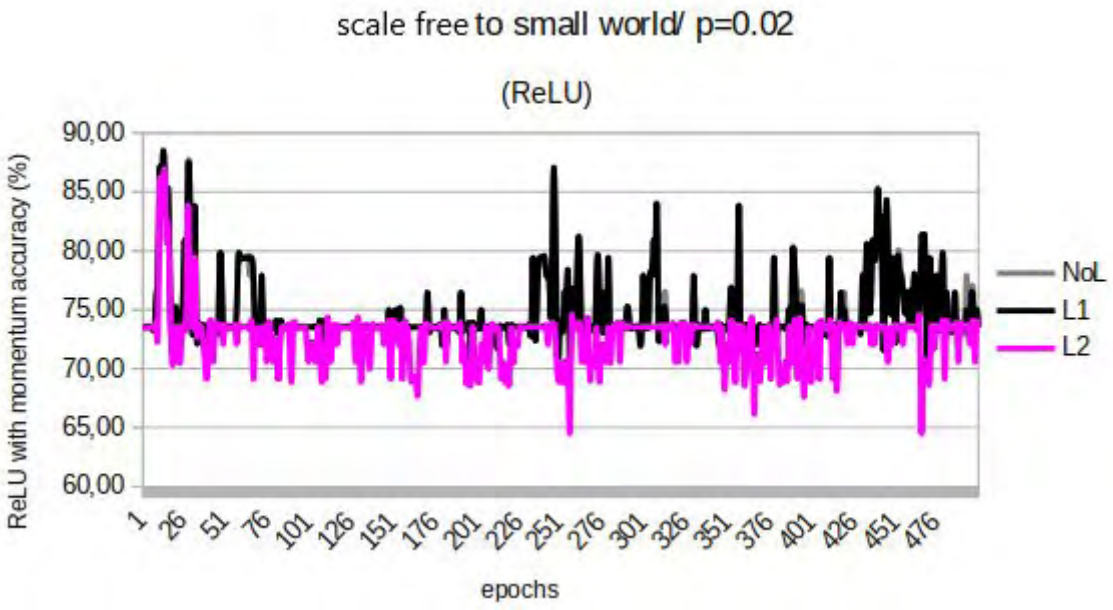


Figure 5.41: Scale Free to Small World accuracy, using ReLU activation function, lung.mat file and p=0.02.

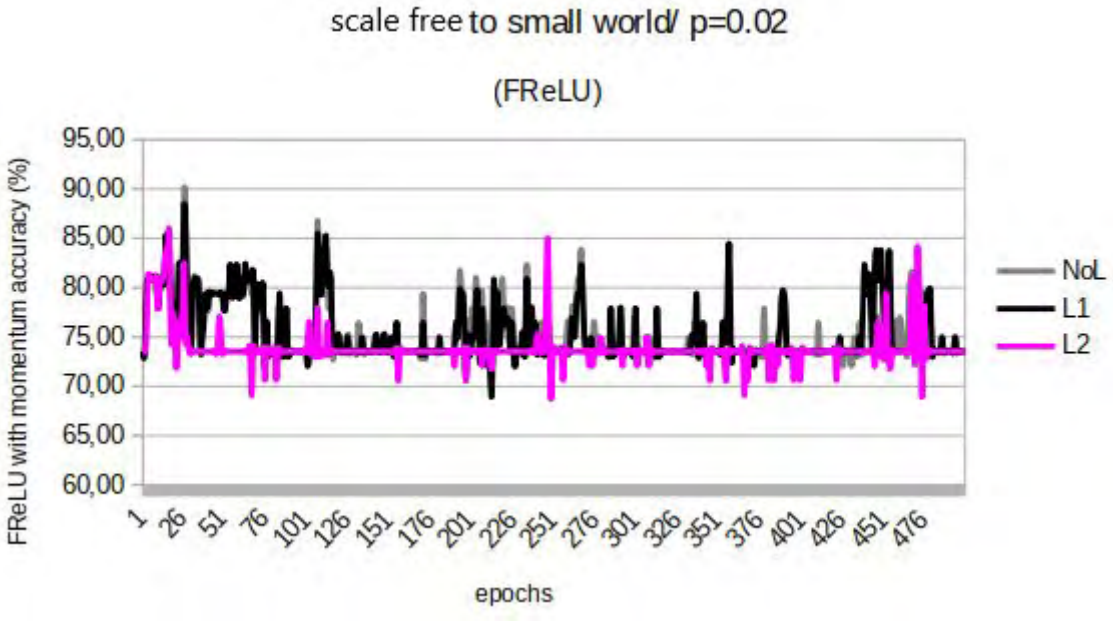


Figure 5.42: Scale Free to Small World accuracy, using FReLU activation function, lung.mat file and p=0.02.

In this case, we create another algorithm, combining scale-free and small-world techniques. Our estimation is that the accuracy is decreased when a network, following a power law degree distribution, transits to a more randomly linked topology. Specifically, both in ReLU and FReLU implementation, NoL and L1 regularized curves are overlapped (same accuracy) and they differ only in execution time. We see that L1 parameter contributes more efficiently to the network when combined with FReLU function (regarding accuracy). This approach has, also, much larger execution time than the others, owing to its great computational complexity.

ReLU	NoL	74.7	4 h 13 mins
ReLU	L2	72.6	4 h 14 mins
ReLU	L1	74.8	4 h 17 mins
FReLU	NoL	75.3	4 h 20 mins
FReLU	L2	73.8	4 h 19 mins
FReLU	L1	75.4	4 h 17 mins

Table 5.22: Statistics of Scale Free to Small World algorithm, using lung.mat file and p=0.02.

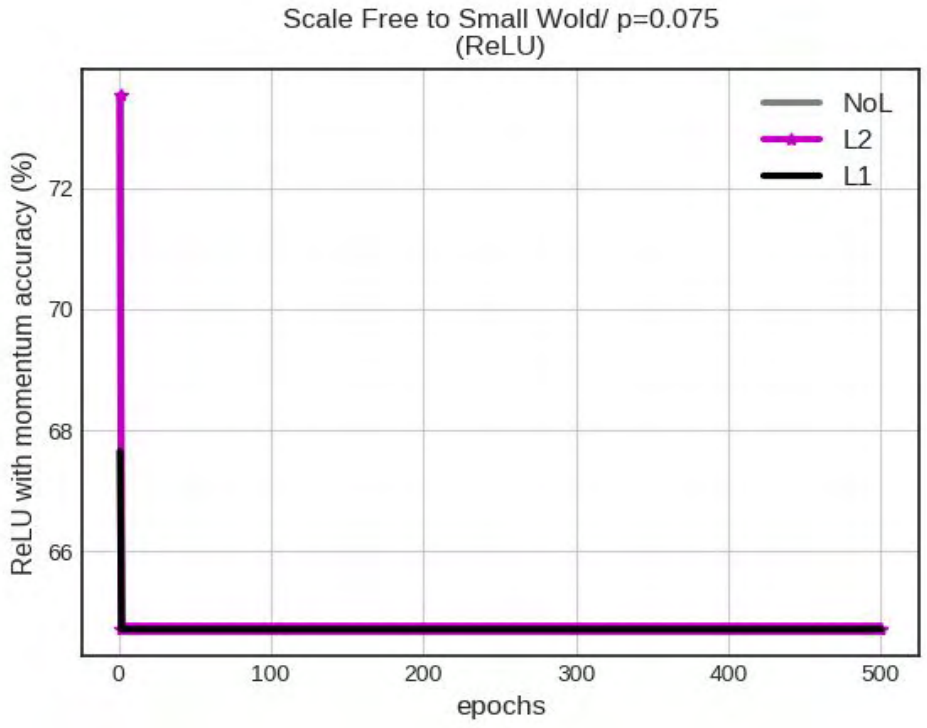


Figure 5.43: Scale Free to Small World accuracy, using ReLU activation function, lung.mat file and p=0.075.

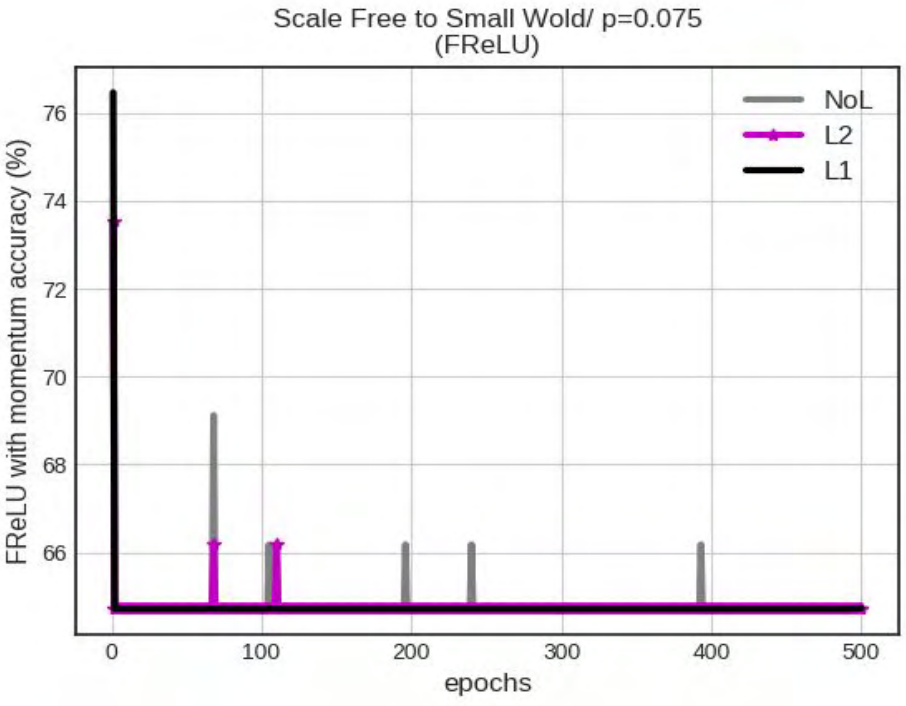


Figure 5.44: Scale Free to Small World accuracy, using FReLU activation function, lung.mat file and p=0.075.

We run the same algorithm, with a larger probability. While the probability becomes larger, the average length becomes smaller, which means more connections between the nodes (density). Hence this algorithm needs more time to make all these computations between the nodes (enormous execution time). In this case, the network, constructed after the training procedure, tends to be more like an exact random graph than the one in Figure 5.43, 5.44. Particularly speaking, it is possible for a node to be connected to a less powerful node, so, in the next epoch, the information, saved in this node, maybe, is not going to be transferred to the next layer, because the links reconnected randomly, and so on.

ReLU	NoL	64.7	6 h 3 mins
ReLU	L2	64.7	6 h 15 mins
ReLU	L1	64.7	6 h 20 mins
FReLU	NoL	64.7	6 h 52 mins
FReLU	L2	64.7	6 h 54 mins
FReLU	L1	64.7	6 h 20 mins

Table 5.23: Statistics of Scale Free to Small World algorithm, using lung.mat file and p=0.075.

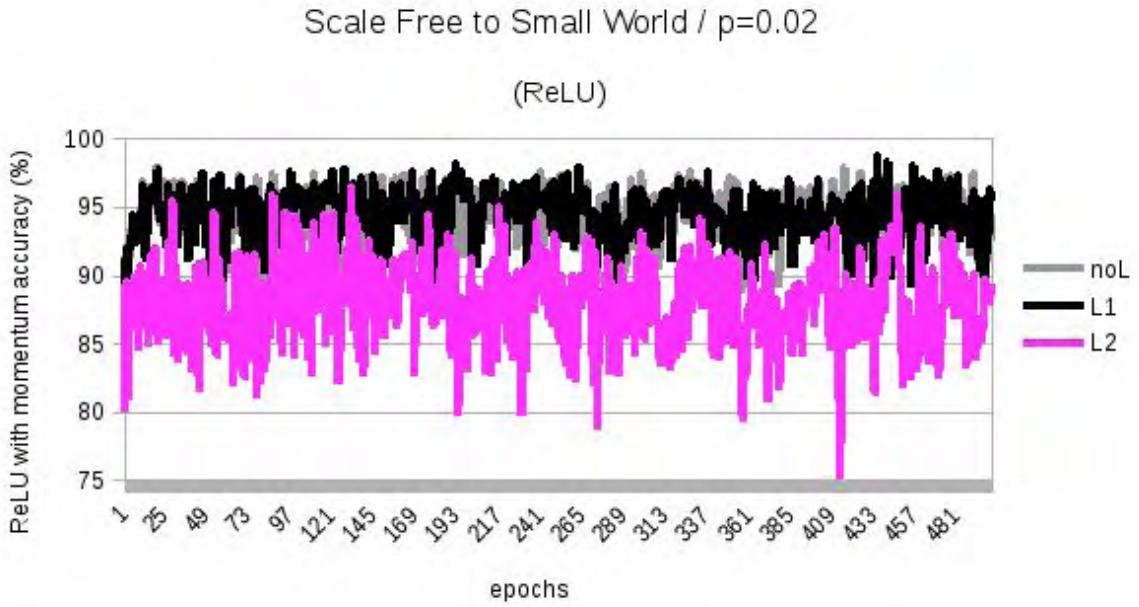


Figure 5.45: Scale Free to Small World accuracy, using ReLU activation function, COIL20.mat file and p=0.02.

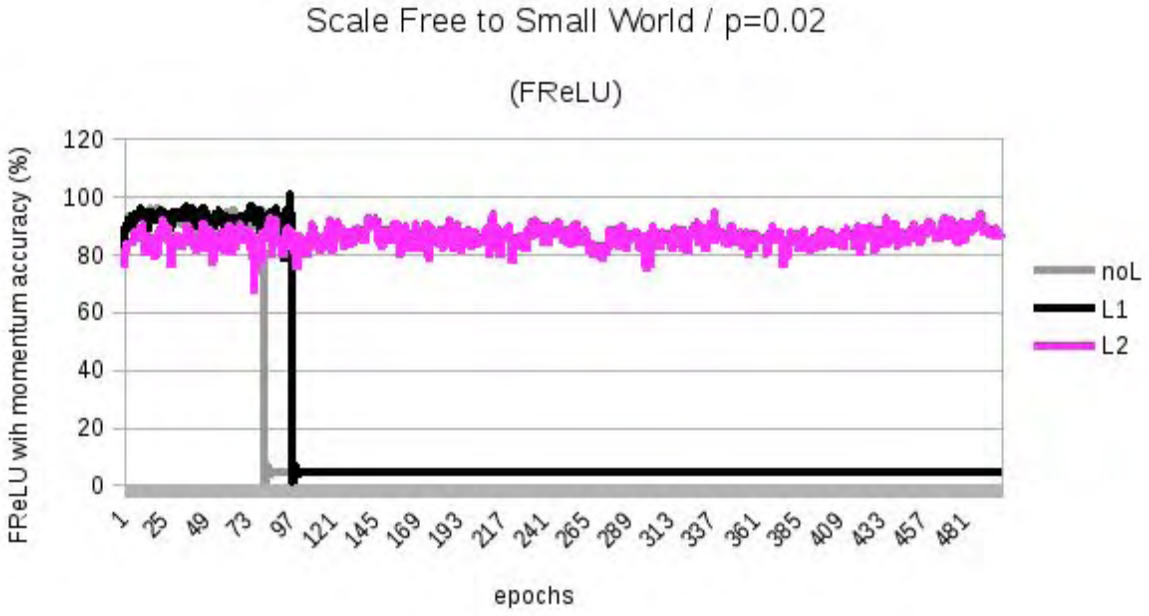


Figure 5.46: Scale Free to Small World accuracy, using FReLU activation function, COIL20.mat file and p=0.02.

This variant of our concept has moderate accuracy and is trained very slowly, when we use biological input data. So, we test it with a different type of input

data (images) and we notice that the accuracy increases (94.36%) and the training time decreases a lot (2 h 9 mins), despite the fact that this dataset has more classes than lung.mat dataset and we expect to see the training time increasing. This is due to the fact that small-world graphs has more random topology and so they need more computational complexity to be reconstructed from a scale-free network. So a file with a lot of features, such as lung.mat, does not help but makes training phase slow. We observe in Figures 5.45 and 5.46 that average accuracy values vary from 87% to 95%. Conversely, in cases of FReLU and NoL or L2 regularizations, the accuracy is very low (about 20%) because of a code warning.

ReLU	NoL	94.36	2 h 9 mins
ReLU	L2	87.98	2 h 13 mins
ReLU	L1	94.35	2 h 49 mins
FReLU	NoL	18.38	2 h 20 mins
FReLU	L2	85.83	2 h 47 mins
FReLU	L1	21.23	3 h 47 mins

Table 5.24: Statistics of Scale Free to Small World algorithm, using COIL20.mat file and p=0.02.

•

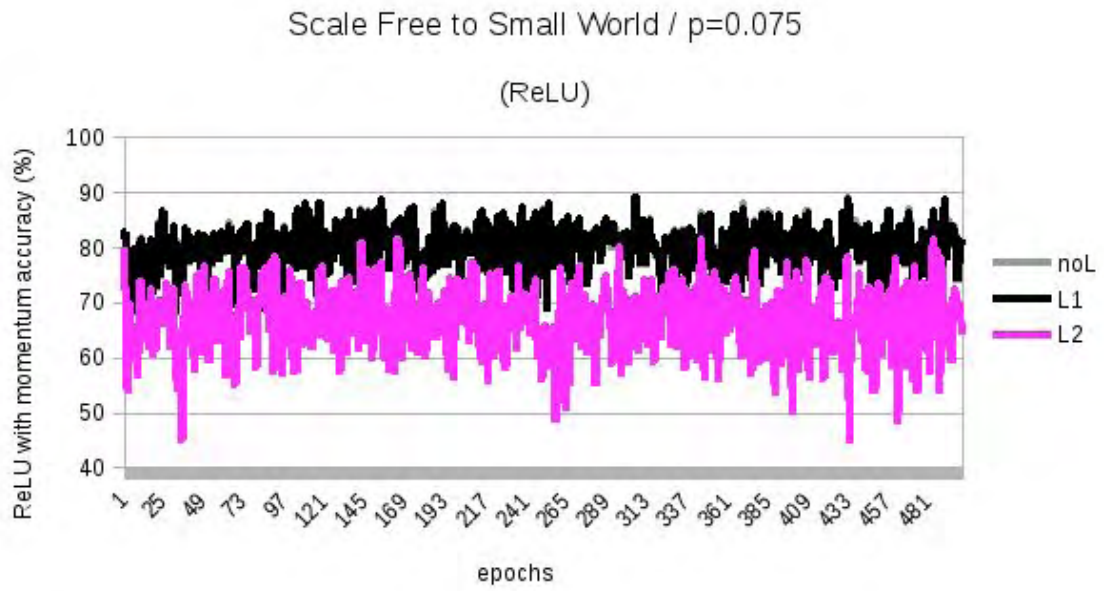


Figure 5.47: Scale Free to Small World accuracy, using ReLU activation function, COIL20.mat file and p=0.075.

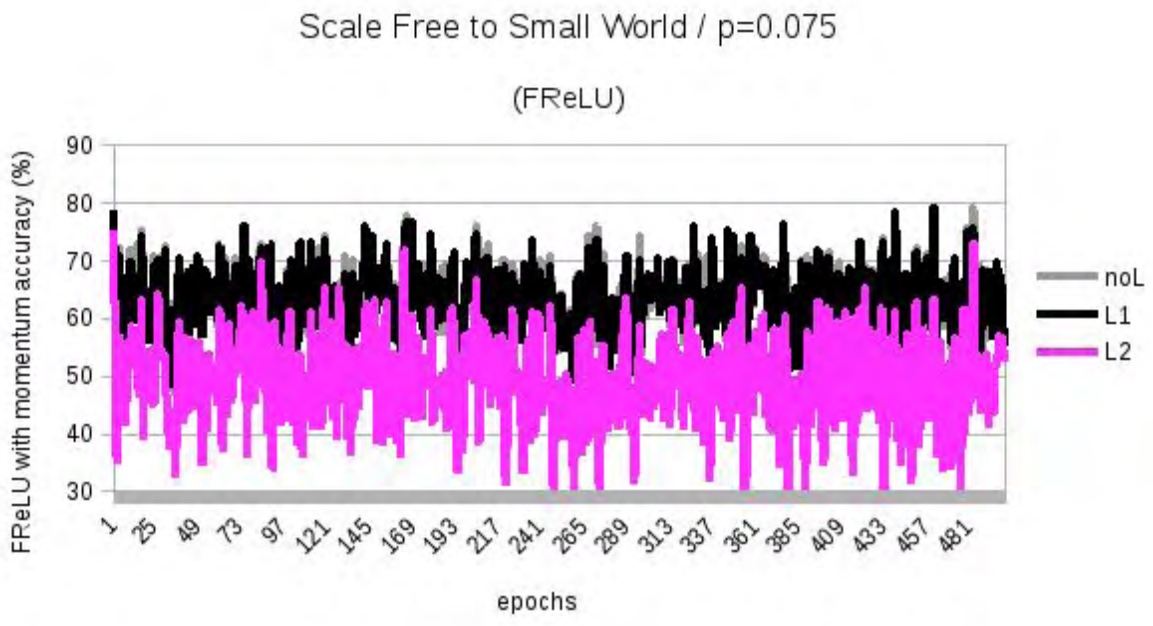


Figure 5.48: Scale Free to Small World accuracy, using FReLU activation function, COIL20.mat file and p=0.075.

We test the algorithm with a larger probability (p = 0.075) and same image dataset. Larger rewiring probability means that the network becomes more dense and so has more connections between the nodes. Thus the average accuracy decreases and the neural network needs more time to be trained because of more

computations (more links and so more weight updates). Although, if we compare the results with these of lung.mat file, in this case the accuracy is better and the time is about 2 hours smaller. Particularly, the algorithm achieves 79.85% accuracy in 4 hours and 3 minutes, when ReLU activation function is used without any regularization. Table 5.25 shows the statistics (average accuracy and training time) in all cases of parameters.

ReLU	NoL	79.85	4 h 3 mins
ReLU	L2	66.79	4 h 9 mins
ReLU	L1	79.85	4 h 30 mins
FReLU	NoL	63.26	4 h 37 mins
FReLU	L2	49.06	4 h 56 mins
FReLU	L1	62.99	5 h 13 mins

Table 5.25: Statistics of Scale Free to Small World algorithm, using COIL20.mat file and p=0.075.

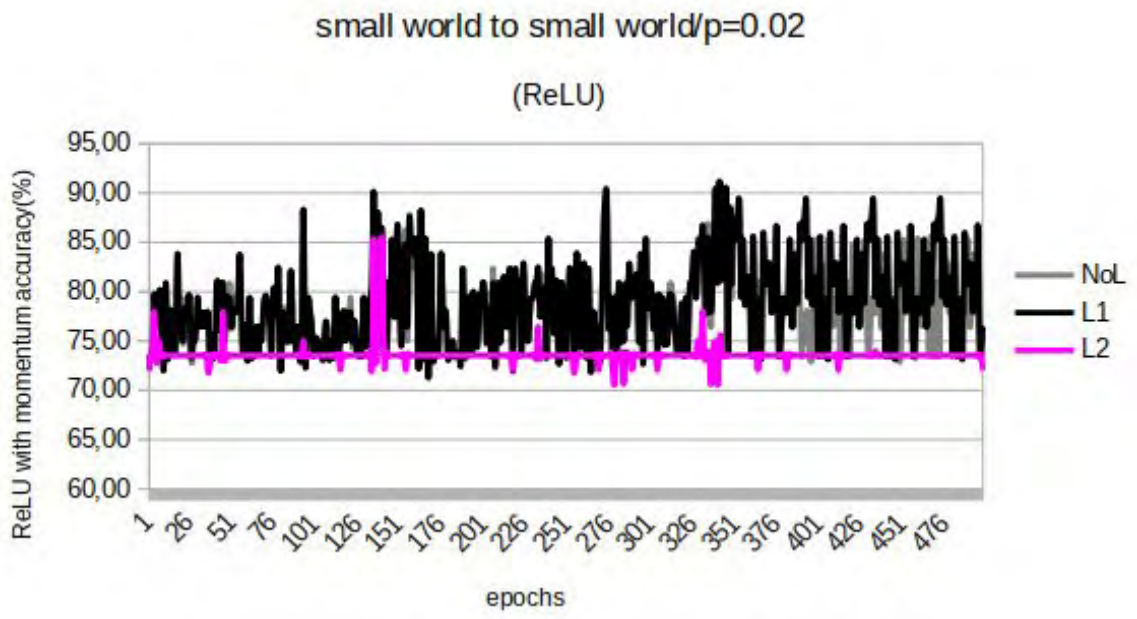


Figure 5.49: Small World to Small World accuracy, using ReLU activation function, lung.mat file and p=0.02.

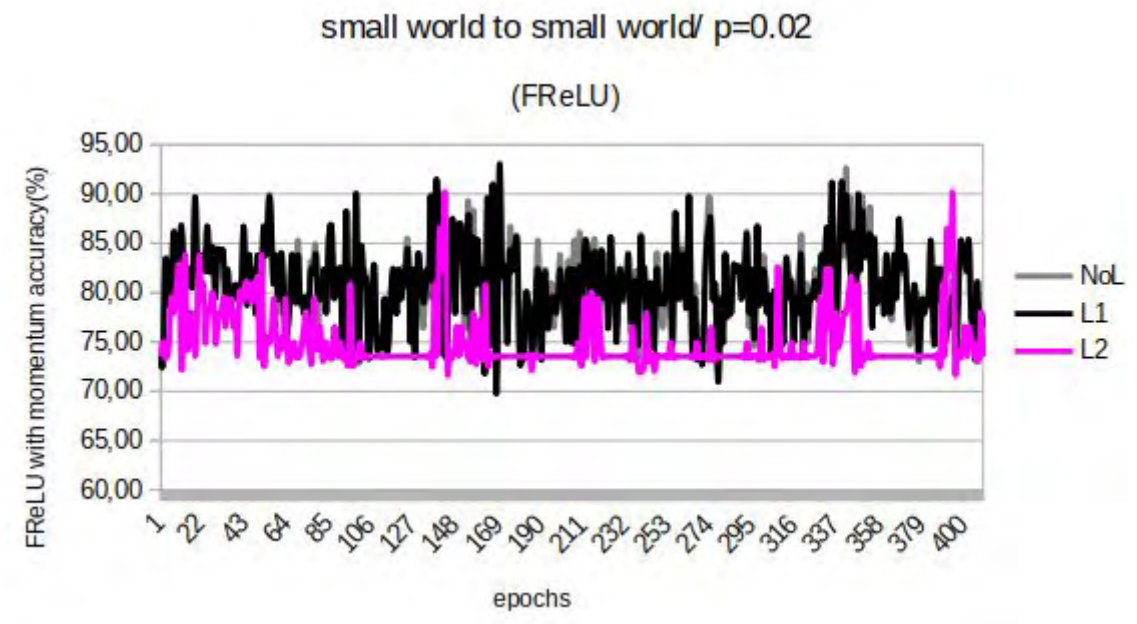


Figure 5.50: Small World to Small World accuracy, using FReLU activation function, lung.mat file and p=0.02.

In this last case, we study the performance of the neural network system in terms of using only the small-world method. Using FReLU activation, algorithm achieves better accuracy (81%) but the problem is that algorithm takes enough time to train the neural network (enormous execution time - because of computational complexity). Also, the rewiring probability is p = 0.02 (small enough) which means the reconstructed network is not random enough. Hence the accuracy is stabilized in high levels of values. Therefore, our estimation, again, is that the less a topology is random, the more efficient the network becomes, in learning and generalizing the data.

ReLU	NoL	78	4 h 9 mins
ReLU	L2	74	4 h 8 mins
ReLU	L1	78	4 h 12 mins
FReLU	NoL	81	4 h 13 mins
FReLU	L2	75	4 h 15 mins
FReLU	L1	81	5 h

Table 5.26: Statistics of Small World to Small World algorithm, using lung.mat file and p=0.02.

•

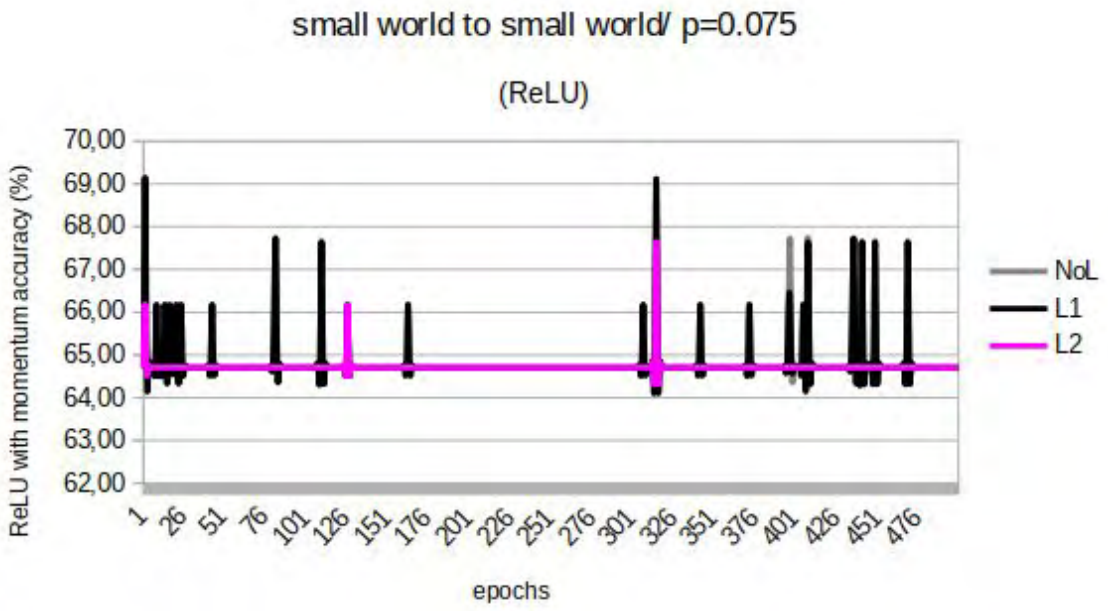


Figure 5.51: Small World to Small World accuracy, using ReLU activation function, lung.mat file and p=0.075.

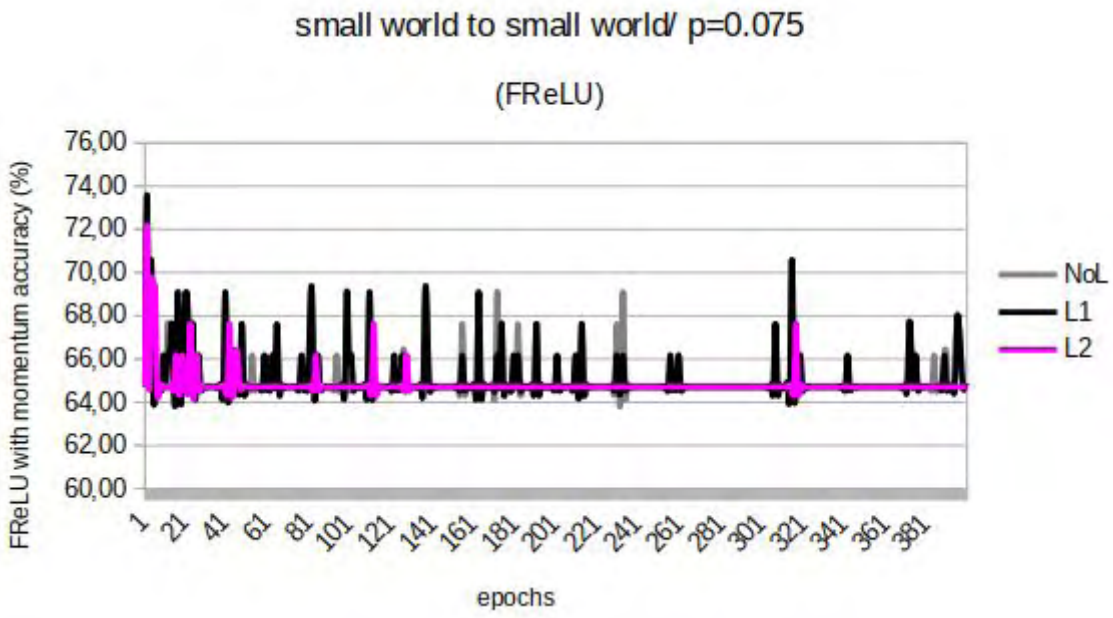


Figure 5.52: Small World to Small World accuracy, using FReLU activation function, lung.mat file and p=0.075.

In contrast to the results in Figure 5.51 and 5.52, in this case we use a much larger probability (smaller clustering coefficient), which makes the graph denser. Not only for its density, but also for its computational complexity, this variant tends to need more training time. Due to its randomness, information is distributed in every possible node (not in the popular ones), meaning that the information is not retained while passing through the epochs. This affects the

network and so as the accuracy which is slow enough (65%). We see, in Figure 5.52 that L1 and NoL regularized curves have a lot of fluctuations while FReLU function is used. Maybe, this is owing to the fact that FReLU provides more capacity than ReLU, which leads the model not to generalize well from its training data to unseen data, regarding the trainable dataset.

ReLU	NoL	64.8	6 h 3 mins
ReLU	L2	64.7	6 h 6 mins
ReLU	L1	64.8	6 h 55 mins
FReLU	NoL	65	6 h 5 mins
FReLU	L2	65	6 h 11 mins
FReLU	L1	65	5 h 50 mins

Table 5.27: Statistics of Small World to Small World algorithm, using lung.mat file and p=0.075.

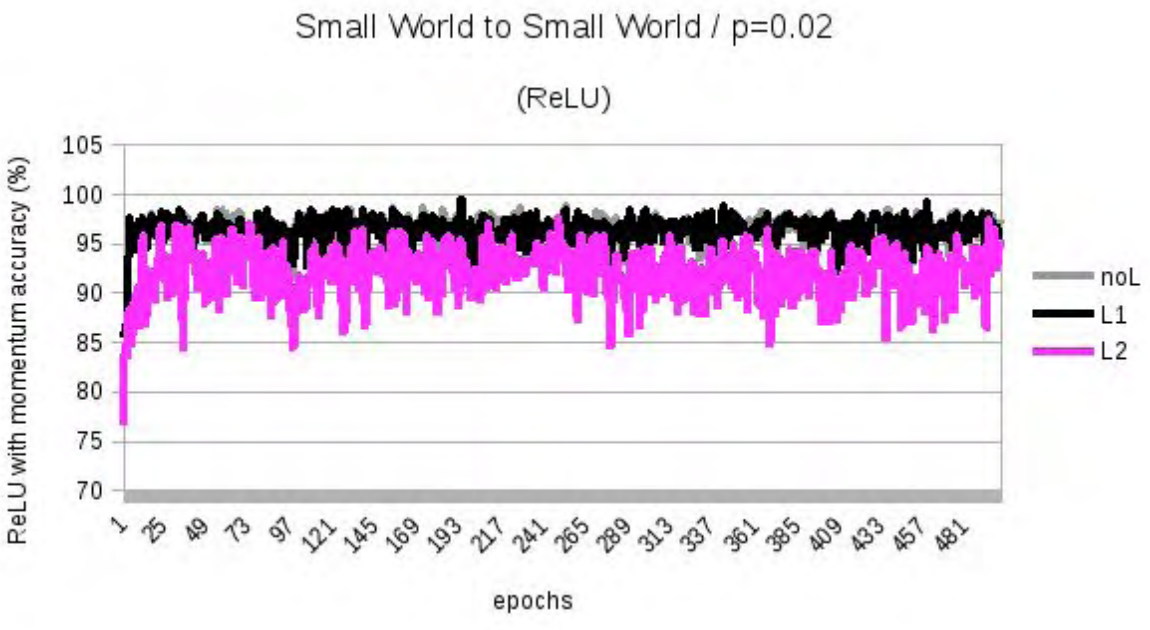
- 

Figure 5.53: Small World to Small World accuracy, using ReLU activation function, COIL20.mat file and p=0.02.

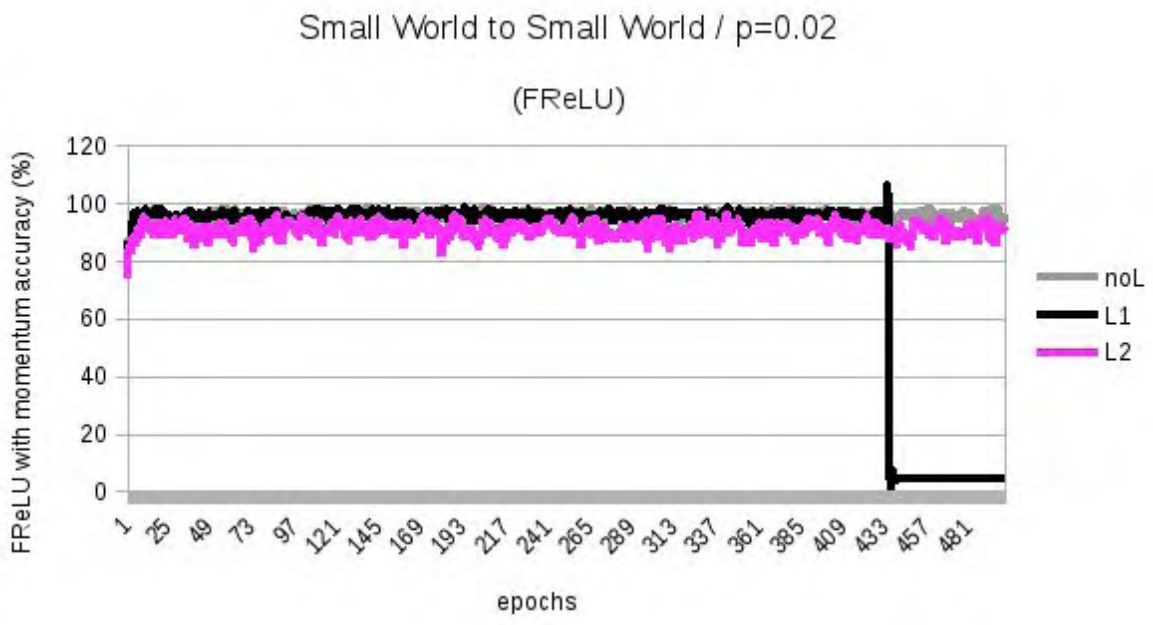


Figure 5.54: Small World to Small World accuracy, using FReLU activation function, COIL20.mat file and p=0.02.

According to the Figures 5.53 and 5.54 and Table 5.28, we can see that this algorithm has better results than the variant of Scale Free to Small World, despite the fact that small-world graphs has more random topology. This is due to the fact that we construct a small-world graph starting by a identical one, which has the same structure, and thus, the neural network does not takes long time to construct it and to be trained. If we compare the results of this case with results on lung.mat file, we can see that algorithm with COIL20.mat file has again better performance, because of large amount of samples in the dataset. Although we expect the training time to be larger than the lung.mat file (greater amount of classes), we observe again that algorithm needs approximately the half training time, as the case of Scale Free to Small World variant (smaller amount of features). The best performance of the algorithm is reached in the combination of ReLU activation function and L1 regularization (96.15% in 2 hours and 44 minutes). Finally, we notice in Figure 5.54 that L1 curve falls off abruptly, due to a code warning (in 440 epoch).

ReLU	NoL	96.14	2 h 8 mins

ReLU	L2	91.96	2 h 8 mins
ReLU	L1	96.15	2 h 44 mins
FReLU	NoL	95.69	2 h 41 mins
FReLU	L2	90.69	2 h 53 mins
FReLU	L1	83.65	3 h 15 mins

Table 5.28: Statistics of Small World to Small World algorithm, using COIL20.mat file and p=0.02.

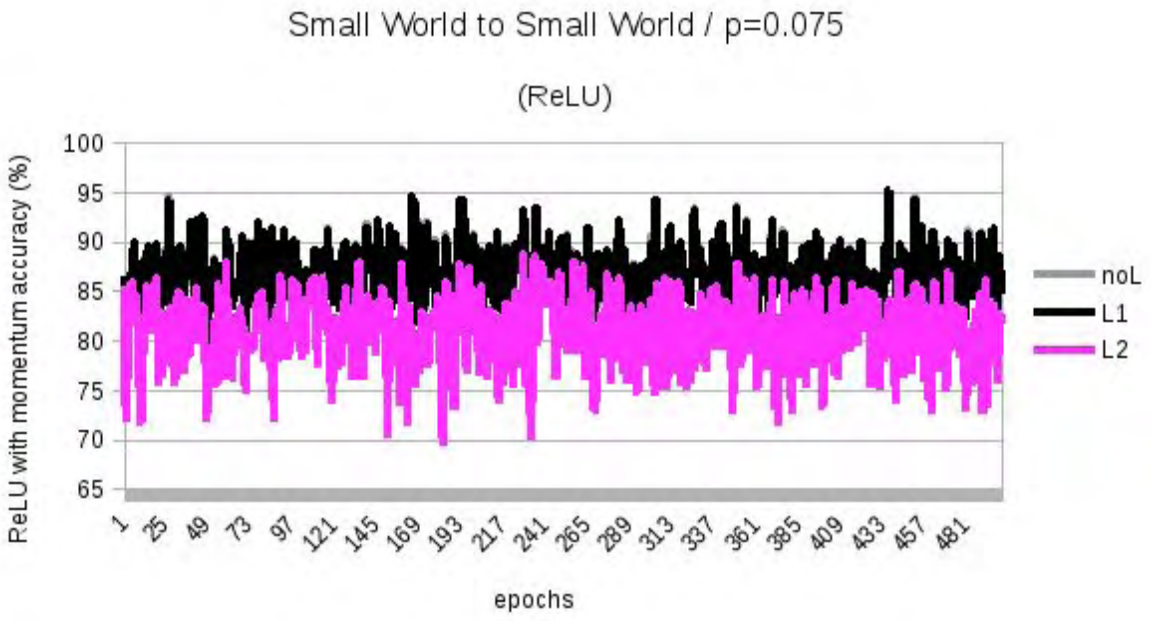
- 

Figure 5.55: Small World to Small World accuracy, using ReLU activation function, COIL20.mat file and p=0.075.

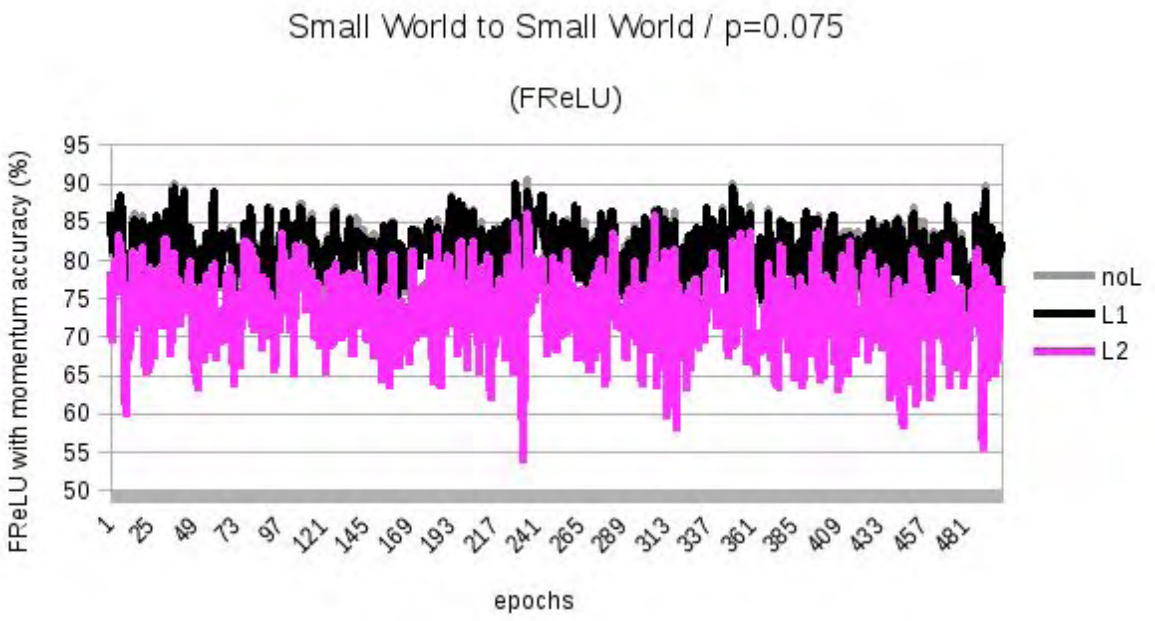


Figure 5.56: Small World to Small World accuracy, using FReLU activation function, COIL20.mat file and p=0.075.

Observing Figures 5.55 and 5.56, we can see that this variant of our concept achieves approximately 80%-87% accuracy in about 3 hours and 30 minutes. Also, we notice that there is not any fluctuation in the graphs which means that the algorithm' s behaviour is stable. Comparing the statistics in Table 5.29 with these in Table 5.27, we conclude that when we use COIL20.mat file we have higher accuracy and less training time, because of smaller amount of features and larger number of instances. The best combination in this algorithm is ReLU transfer function and without any regularization (86.58% in 3 hours and 29 minutes). The rewiring probability of value 0.075 leads the algorithm to a lower performance, because the network does not have such a structured topology, while probability increasing.

ReLU	NoL	86.58	3 h 29 mins
ReLU	L2	80.81	3 h 36 mins
ReLU	L1	86.57	4 h 5 mins
FReLU	NoL	80.67	4 h 5 mins
FReLU	L2	73	4 h 11 mins

FReLU	L1	80.63	4 h 40 mins

Table 5.29: Statistics of Small World to Small World algorithm, using COIL20.mat file and p=0.075.

In this chapter, we analyze the network behavior, including all the techniques we proposed and all the datasets, mentioned in the *Evaluation Settings* section. Our concept is based on SET algorithm (described in chapter 4), which starts from a randomly-structured topology and via a randomization procedure, sparsifies the current network and produces a kind of scale-free attributed structured topology. The results are satisfying and the time needed is about 40 minutes in the best case. It is a good technique which emphasizes in links, between the nodes, that have weights which can really reinforce the important information. However, the network randomness doesn't help the network distribute the information suitably (due to zero-clustering). Maintaining the same method for network sparsification (remove links close to zero) and in order to improve the network structure for better information distribution, we propose, in the first place, the Scale Free to SET algorithm. An advantage of our algorithm is that the training procedure takes half time than the corresponding one in SET, due to the fact that we start from strictly constructed network and end up producing a same one, proving that the more structured a network is, the better the information is managed. Furthermore, the accuracy that algorithm achieves is similar to the SET one, as Figure 5.57 shows for different files, which makes this variant an improvement regarding competitor's code (SET). It is interesting how the structured topology affects the network performance. Hence, our second variant of concept is the Scale Free to Scale Free algorithm, which differs from Scale to SET one, only in the produced network, which follows exactly a power-low degree distribution. Observing the graphs illustration in 5.1 section, we can see that the accuracy is much higher than the accuracy from both SET and Scale to SET, owing to the better network construction (presence of hubs - high clustering). As it is obvious, its training time is larger than the corresponding one in Scale to SET (due to more calculations in produced net) but is smaller than SET (in which starting from a complete random topology, it might take too much time to reach a structured, scale-free or small-world, topology). Bearing in mind that a strictly structured network is beneficial for both accuracy and time and being simultaneously, inspired by network science theory, we find interesting to explore the performance of small-world networks. So, we implement Scale Free to Small World algorithm. The performance of this variant isn't as good as we expected to be. The accuracy

decreases in respect with the previous algorithms, especially, in cases where rewiring probability of small-world method isn't small enough. Maybe, this occurs because of its more random reconstruction after training phase. Regarding the time, the learning part takes more time, because of not only for its density (smaller clustering coefficient), but also for its computational complexity. Finally, our last proposal is the one which includes a transition, being from a less randomly constructed network to another similar one. Not only for the accuracy, but also for its training time, this implementation is disappointing. It takes approximately the same time as the previous implementation, for the network to be trained and the accuracy is very low (as rewiring probability increases - network tends to be more random). In conclusion, we can say that in cases where high clustering exists (more strictly structured net like scale-free ones), then we have more efficient results. The results of every algorithm we constructed, are depicted in Figure 5.57 and the time needed for the training phase to be accomplished is depicted in Figure 5.58 (based on best accuracy).

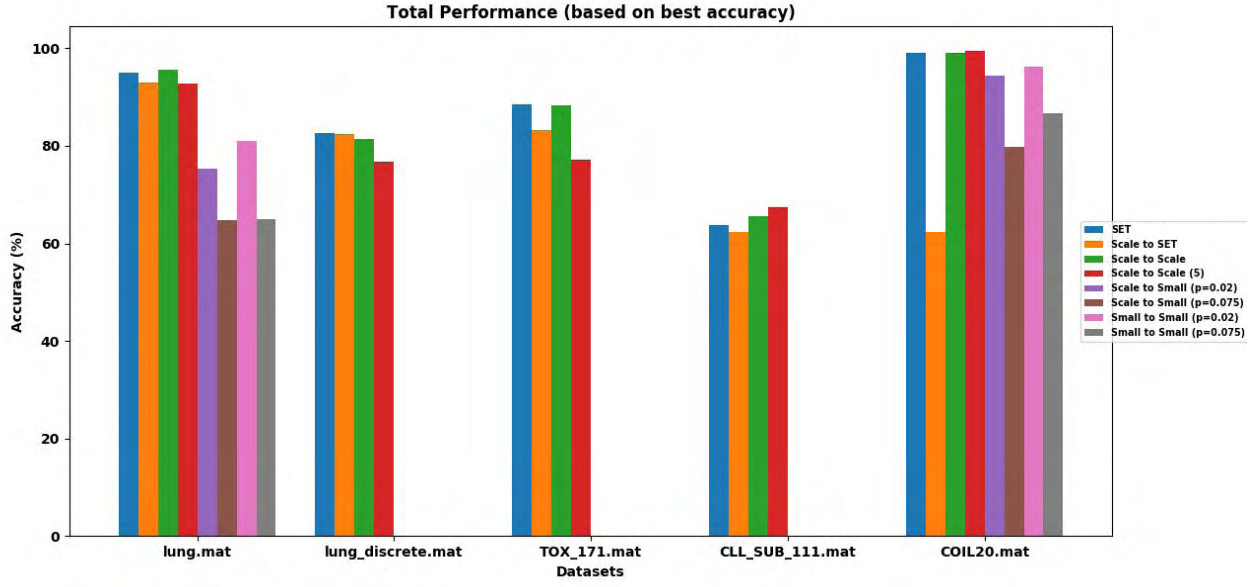


Figure 5.57: Accuracy evaluation of five algorithms, including SET code, using five different datasets

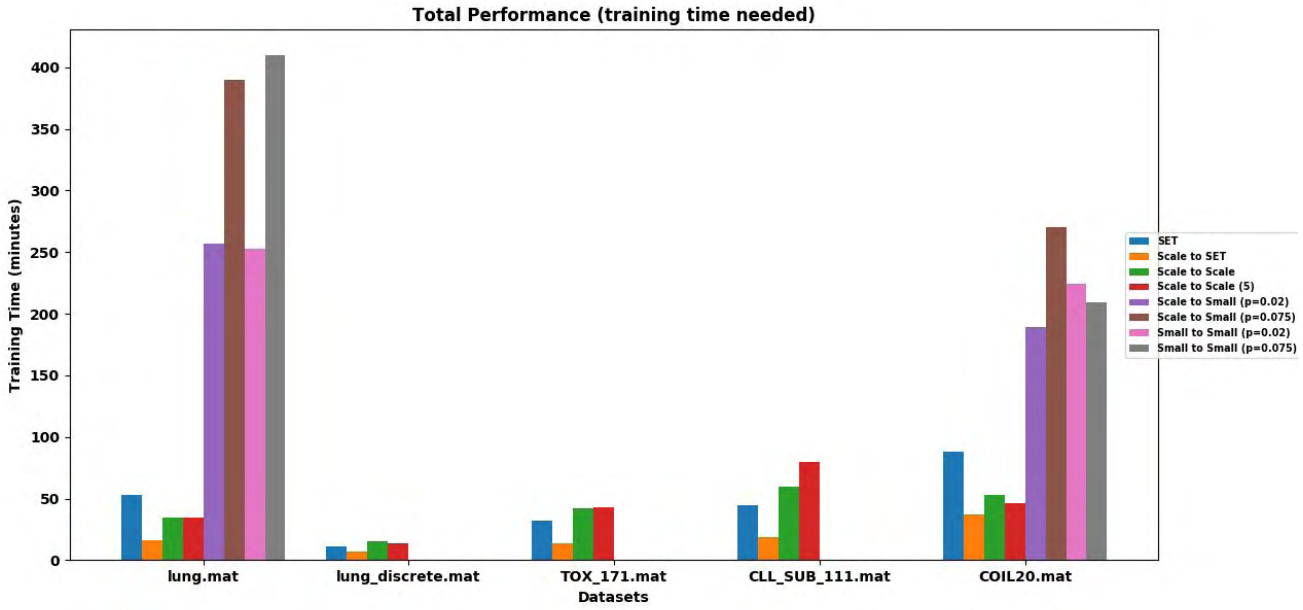


Figure 5.58: Time evaluation (in minutes) of five algorithms, including SET code, using five different datasets

The tremendous success of deep learning has brought neural networks at the forefront of machine learning research and development. Due to the large size of a neural network – in number of neurons and in number of hidden layers – training the network in relative short time is a challenge. Various families of methods have been developed for accelerating neural training during the past thirty years. We focus here in the family of methods that are based on linkage sparsification, i.e., instead of having fully connected bipartite neural topologies, we reduce the number of connections in an algorithmic (or in a random) way. In particular, we employ concepts developed in the realm of network science, in order to sparsify the neural network and thus reduce drastically the number of trainable variables and achieve training acceleration. We base our motivation on observations in real neural networks whose actual topology is scale-free or small-world. We designed algorithms that start from a particular structured, but not fully connected bipartite topology, and end up with another structured topology. Here, in this first investigation we experimented with scale-free and small-world topologies either as starting or final topologies. We evaluated the algorithms performance on a moderate size neural network in a publicly available dataset, and examined their classification accuracy and training time. We concluded that the proposed techniques are able to reap performance gains, achieving high accuracy with short training time. The "champion" algorithm was the one that produced scale-free topologies starting from scale-free topologies. Intuitively this is expected, since only a handful of connections carry most of the weight even in fully connected topologies. Our results are consistent with recent but different types of approach [37][38] to the problem of neural training acceleration.

[1] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, Antonio Liotta. 2018. "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science". NATURE COMMUNICATIONS (June 19, 2018).

[2] Zhongzhi Zhang, Shuigeng Zhou , Lichao Chen. 2007. "Evolving pseudofractal networks" (september 2007).

[3] Xiao Fan Wang, Guanrong Chen. 2003. "Complex Networks: Small-World, Scale-Free and Beyond", IEEE circuits and systems magazine (2003).

[4] Suo Qiu, Xiangmin Xu, Bolun Cai. 2018. "FReLU: Flexible Rectified Linear Units for
Improving Convolutional Neural Networks". (january 2018).

[5] Xiaojie Jin, Chunyan Xu, Jiashi Feng, Yunchao Wei, Junjun Xiong, Shuicheng Yan. 2015. "Deep Learning with S-shaped Rectified Linear Activation Units ". (December 2015).

[6] D. Volchenkov, Ph. Blanchard. 2018. "AN ALGORITHM GENERATING SCALE FREE GRAPHS". (October 23, 2018).

[7] Insoo Sohn. 2017. "Small-World and Scale-Free Network Models for IoT Systems". Hindawi, Mobile Information Systems (January 30, 2017).

[8] Groth, David; Toby Skandier (2005). Network+ Study Guide, Fourth Edition. Sybex, Inc. ISBN 0-7821-4406-3.

[9] Albert Barabasi, Eric Bonabeau,2003. "Scale-Free Networks". SCIENTIFIC AMERICAN (May, 2003).

[10] Qawi K. Telesford, Karen E. Joyce, Satoru Hayasaka, Jonathan H. Burdette, Paul J. Laurienti, 2011. "The Ubiquity of Small-World Networks". (2011).

[11] Theoden I. Netoff, Robert Clewley, Scott Arno, Tara Keck, John A. White. 2004. "Epilepsy in Small-World Networks". The Journal of Neuroscience (September 15, 2004).

[12] Ye Hoon Lee, Insoo Sohn. 2017. "Reconstructing Damaged Complex Networks Based on Neural Networks". Symmetry 2017, 9, 310 (December 9, 2017).

[13] Konstantinos Diamantaras. "Texnita Neurwnika Diktua". Klidarithmos, 2007.

[14] S. Havlin, N. A. M. Araujo, S. V. Buldyrev, C. S. Dias, R. Parshani, G. Paul, H. E. Stanley, 2010. "Catastrophic Cascade of Failures in Interdependent Networks". (December, 2010).

[15] Nicholas Jarman, Erik Steur, Chris Trengove, Ivan Y. Tyukin & Cees van Leeuwen. 2017. "Self-organisation of small-world networks by adaptive rewiring in response to graph diffusion". (October 13, 2017)

[16] M. E. J. Newman, D. J. Watts, S. H. Strogatz, 2002. "Random graph models of social networks".Newman et al, PNAS ,vol. 99 suppl. 1 (February 19, 2002).

[17] Tom Dietterich, 1995. "Overtting and Undercomputing in Machine Learning".ACM Computing Surveys (CSUR),Volume 27 Issue 3 (September, 1995).

[18] Vanessa Isabell Jurtz, Alexander Rosenberg Johansen, Morten Nielsen, Jose Juan Almagro Armenteros, Henrik Nielsen, Casper Kaae Sønderby, Ole Winther, Søren Kaae Sønderby. 2017. "An introduction to deep learning on biological sequence data: examples and solutions" (August 23, 2017).

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov,2014."Dropout: A Simple Way to Prevent Neural Networks from Overfitting". Journal of Machine Learning Research 15 (2014) 1929-1958 (June, 2014).

[20] D. Randall Wilsona, Tony R. Martinez. 2003. "The general inefficiency of batch training for gradient descent learning" (April 8, 2003).

[21] Andrew Y. Ng, 2004. "Feature selection, L1 vs. L2 regularization, and rotational invariance". 21st International Conference on Machine Learning, Banff, Canada (2004).

[22] M Moreira and E Fiesl. 1995. "Neural Networks with Adaptive Learning Rate and Momentum Terms" (October, 1995).

[23] Ning Qian. "On the Momentum Term in Gradient Descent Learning Algorithms". Neural Networks archive, Volume 12 Issue 1 (January, 1999).

[24] Yonina C. Eldar. 2006. "Mean-Squared Error Sampling and Reconstruction in the Presence of Noise" (December 12, 2006).

[25] Albert-Lazlo Barabasi, "NETWORK SCIENCE: THE SCALE-FREE PROPERTY".

[26] Michael Small, Xiaoke Xu, Jin Zhou, Jie Zhang, Junfeng Sun, Jun-an Lu, 2008. "Scale-free networks which are highly assortative but not small world". The American Physical Society, PHYSICAL REVIEW E 77, 066112 (June 20, 2008).

[27] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning". (November 8, 2018)

[28] Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola. 2019. "Dive into Deep Learning". (May 19, 2019).

[29] Jamal M. Nazzal, Ibrahim M. El-Emary and Salam A. Najim. 2008. "Multilayer Perceptron Neural Network (MLPs) For Analyzing the Properties of Jordan Oil Shale". World Applied Sciences Journal 5 (5): 546-552, 2008, ISSN 1818-4952.

[30] Franco Scarselli, Ah Chung Tsoi,1998. "Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results". Neural Networks, Vol. 11, No. 1, pp. 15–37, 1998.

[31] Syrine Ben Driss, M Soua, Rostom Kachouri, Mohamed Akil, 2017. "A comparison study between MLP and Convolutional Neural Network models for character recognition", SPIE Conference on Real-Time Image and Video Processing, Apr 2017, Anaheim, CA, United States. ff10.1117/12.2262589ff. ffhal-01525504f.

[32] Nitin Malik. 2005. "Artificial Neural Networks and their Applications". (June, 2005).

[33] Alex Pappachen James,Sima Dimitrijev, 2012."Feature selection using nearest attributes".(January 28, 2012)

[34] Ricard V. Sole, Sergi Valverde, 2004. "Information Theory of Complex Networks: on evolution and architectural constraints", Lecture Notes in Physics 207:189-207, (2004, August 19).

[35] L. A. N. Amaral, A. Scala, M. Barthelemy, H. E. Stanley,2000. "Classes of small-world networks", Proc Natl Acad Sci U S A, (2000, October 10).

[36] Hans-Dieter Wehle. 2017. "Machine Learning, Deep Learning, and AI: What's the Difference?". (July, 2017).

[37] J. Frankle, M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks", *Proceedings of the Inernational Conference on Learning Representations (ICLR)*, 2019.

[38] X. Sun, X. Ren, S. Ma, B. Wei, W. Li, J. Xu, H. Wang, Y. Zhang, "Training simplification and model simplification for deep learning: A minimal effort back propagation method", *IEEE Transactions on Kowledge and Data Engineering*, to appear, 2019.

[39] Tianyi Liu, Shuangsang Fang, Yuehui Zhao, Peng Wang, Jun Zhan. "Implementation of Training Convolutional Neural Networks", arXiv:1506.01195 June 2015.

[40] M. Kukacka. "Overview of Deep Neural Networks", *Neural Networks*, Vol 61, pp 85-117, 2012.

[41] Patrick Glauner. "Comparison of Training Methods for Deep Neural Networks", Neural Networks, Vol 7, No. 1, pp. 1-11, 1994.

[42] lung.mat dataset information. Available [Online]: https://archive.ics.uci.edu/ml/datasets/lung+cancer , accessed May 31,2019.

[43] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. "Recent Advances in Recurrent Neural Networks", arXiv:1801.01078, February 2018.

[44] Mikael Boden. "A guide to recurrent neural networks and backpropagation", *School of Information Science, Computer and Electrical Engineering Halmstad University*, December 2001.

[46] Robert Kowalski, "Artificial Intelligence and Human Thinking", Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, June 8, 2011.

[47] Daniel L. Ly, Volodymyr Paprotski, Danny Yen,"Neural Networks on GPUs:

Restricted Boltzmann Machines". University of Toronto, 2009.

[48] Daniel Strigl, Klaus Kofler and Stefan Podlipnig, "Performance and Scalability of GPU-based Convolutional Neural Networks", *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP 2010, Pisa, Italy, (*February 17-19, 2010).

[49] The Khronos Group, "OpenCL Overview". Available [Online]: http://www.khronos.org/opencl , (August 2009), accessed May,2019.

[50] C. Boyd, "DirectX 11 Compute Shader," in The 35th Int. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2008). Available [Online]: http://s08.idav.ucdavis.edu/boyd-dx11-compute-shader.pdf ,accessed May, 2019.

[51] E. Bullmore, O. Sporns, "Complex brain networks: Graph theoretical analysis of structural and functional systems", *Nature Reviews on Neuroscience*, vol. 10, pp. 186–198, 2009.

[52] D. Katsaros, N. Dimokas, L. Tassiulas, "Social network analysis concepts in the design of wireless ad hoc network protocols", IEEE Network magazine, vol. 24, no. 6, 2010.

[53] D. Katsaros, G. Pallis, K. Stamos, A. Vakali, A. Sidiropoulos, Y. Manolopoulos, "CDNs content outsourcing via generalized communities", IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 1, pp. 137-151, 2009.

[54] P. Basaras, D. Katsaros, L. Tassiulas, "Detecting influential spreaders in complex, dynamic networks", IEEE Computer magazine, vol. 46, no. 4, pp. 26-31, 2013.

[55] A.L.Barabasi, Network Science, Cambridge University Press, 2016.

[56] Diederik P. Kingma, Jimmy Lei Ba. "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION", arXiv:1412.6980, 3rd International Conference for Learning Representations, San Diego, 2015.

[57] Sebastian Bock, Josef Goppold, Martin Weiß. "An improvement of the convergence proof of the ADAM-Optimizer", *Ostbayerische Technische Hochschule (OTH) Regensburg*, Germany, arXiv:1804.10587, 2018.

[58] Matthew D. Zeiler. "ADADELTA: AN ADAPTIVE LEARNING RATE METHOD", Google Inc., USA New York University, USA, arXiv:1212.5701v1, 2012.

[59] Ravagnani Alberto, (2017). "Duality of codes supported on regular lattices, with an application to enumerative combinatorics". Designs, Codes and Cryptography, 86(9), 2035–2063. doi:10.1007/s10623-017-0436-3

[60] Math Insight, Small world networks. Available [Online]: https://mathinsight.org/small_world_network ,accessed May, 2019.

[61] Numpy Routines. Available [Online]: https://docs.scipy.org/doc/numpy/reference/routines.html ,accessed May, 2019.

[62] M.T. Hagan, M. Menhaj, "Training feedforward networks with the Marquardt algorithm," IEEE Transactions on Neural Networks, vol. 5, no. 6, 1994.

[63] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks", IEE Proceedings, vol. 139, no. 3, pp. 301-310, 1992.

[64] A. Mokhtari, A. Ribeiro, "Global convergence of online limited memory BFGS", Journal of Machine Learning Research, vol. 16, pp. 3151–3181, 2015.

[65] S.J. Reddi, S. Kale, S. Kumar, "On the convergence of Adam and beyond", Proceedings of the Inernational Conference on Learning Representations (ICLR), 2018.

[66] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," Neural Networks, vol. 1, no. 4, pp. 295–308, 1988.

[67] T. Tollenaere, "SuperSAB: Fast adaptive back propagation with good scaling properties", Neural Networks, vol. 3, no. 5, pp. 561–573, 1990.

[68] Iman Sadeghkhani, Abbas Ketabi , and Rene Feuillet, 2013, "Delta-Bar-Delta and Directed Random Search Algorithms Application to Reduce Transformer Switching Overvoltages ", International Journal on Electrical Engineering and Informatics - Volume 5, Number 1, (March 2013).

[69] Mohammed A. Otair and Walid A. Salameh, 2005. "AN ENHANCED VERSION OF DELTA-BAR-DELTA ALGORITHM", The International Conference on Information Technology, (July 2005).

[70] Albert-László Barabási, Réka Albert, 1999."Emergence of Scaling in Random

Networks", SCIENCE VOL 286 (OCTOBER 15, 1999).

[71] Z. Zainuddin, N. Mahat, and Y. Abu Hassan. "Improving the Convergence of the Backpropagation Algorithm Using Local Adaptive Techniques", World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Vol:1, No:1, 2007.

[72] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", Journal of Machine Learning Research, vol. 15, no. 1 pp. 1929–1958, 2014.

[73] S. Cai, J. Gao, M. Zhang, W. Wang, G. Chen, B.C. Ooi, "Effective and efficient dropout for deep convolutional neural networks", arxiv.org 1904.03392, 2019.

[74] X. Sun, X. Ren, S. Ma, H. Wang, "meProp: Sparsified back propagation for accelerted deep learning with reduced overfitting", Proceedings of the International Conference on Machine Learning (PMLR), 2017.

[75] X. Sun, X. Ren, S. Ma, B. Wei, W. Li, J. Xu, H. Wang, Y. Zhang, "Training simplification and model simplification for deep learning: A minimal effort back propagation method", IEEE Transactions on Kowledge and Dta Engineering, to appear, 2019.

[76] Xu Sun, Xuancheng Ren, Shuming Ma, Houfeng Wang, 2019. "meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting", 34th International Conference on Machine Learning (ICML 2017), (Mar 11,2019).

[77] S. Dey, D. Chen, Z. Li, S. Kundu, K.-W. Huang, K.M. Chugg, P.A. Beerel, "A highly parallel FPGA implementation of sparse neural network training", arxiv.org 1806.01087, 2018.

[78] A. Zlateski, K. Lee, H.S. Seung, "Scalable training of 3D convolutional networks on multi- and many-cores", Journal of Parallel and Distributed Computing, vol. 106, 2017.

[79] N.P. Jouppi, C. Young, N. Patil, D. Patterson, "Domain specific architecture for deep neural networks", Communications of the ACM, vol. 61, no. 9, pp. 50-59, 2018.

[80] Yufeng Hao. "A General Neural Network Hardware Architecture on FPGA", Computer Vision and Pattern Recognition, arXiv:1711.05860, 2017.

[81] Norman P. Jouppi, Cliff Young, Nishant Patil, etc., 2017. "In-Datacenter Performance Analysis of a Tensor Processing Unit", 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada (June 26, 2017).

[82] "RBF Kernel Principal Component Analysis". Available [Online]: http://rasbt.github.io/mlxtend/user_guide/feature_extraction/RBFKernelPCA/,accessed June 2019.

[83] Haider Khalaf Allamy, "METHODS TO AVOID OVERFITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING (COMPARATIVE STUDY)", *Computer Science, Communication and Instrumentation Devices* , Kochi, India (December 27, 2014).

[84] "Top 15 Deep Learning applications that will rule the world in 2018 and beyond". Available [Online]: https://medium.com/@vratulmittal/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-and-beyond-7c6130c43b01, accessed June 2019.

[85] "Technology and human rights", *OpenGlobalRights ,Series co-sponsored with Business & Human Rights Resource Centre and University of Washington Rule of Law Initiative*. Available [Online]: https://www.openglobalrights.org/technology/ ,accessed June 2019.

[86] "Top 10 Applications of Deep Learning". Available [Online]: https://www.greatlearning.in/blog/top-10-applications-of-deep-learning/ ,accessed June 2019.

[87] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell, 2017. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description", IEEE Transactions on Pattern Analysis and Machine Intelligence archive, Volume 39 Issue 4, Page 677-691, (April, 2017).

[88] Andrej Karpathy, Li Fei-Fei, 2017. "Deep Visual-Semantic Alignments for Generating Image Descriptions", IEEE Transactions on Pattern Analysis and Machine Intelligence archive, Volume 39 Issue 4, Page 664-676, (April, 2017).