Applied Intelligence (2023) 53:14102–14127
https://doi.org/10.1007/s10489-022-04195-8

Model reduction of feed forward neural networks for resource-constrained devices

Evangelia Fragkou[1] · Marianna Koultouki[1] · Dimitrios Katsaros[1]

Accepted: 21 September 2022 / Published online: 21 October 2022

Abstract
Multilayer neural architectures with a complete bipartite topology have very high training time and memory requirements. Solid evidence suggests that not every connection contributes to the performance; thus, network sparsification has emerged. We get inspiration from the topology of real biological neural networks which are scale-free. We depart from the usual complete bipartite topology among layers, and instead we start from structured sparse topologies known in network science, e.g., scale-free and end up again in a structured sparse topology, e.g., scale-free. Moreover, we apply smart link rewiring methods to construct these sparse topologies. Thus, the number of trainable parameters is reduced, with a direct impact on lowering training time and a direct beneficial result in reducing memory requirements. We design several variants of our concept (SF2SFrand, SF2SFba, SF2SF5, SF2SW, and SW2SW, respectively) by considering the neural network topology as a Scale-Free or Small-World one in every case. We conduct experiments by cutting and stipulating the replacing method of the 30% of the linkages on the network in every epoch. Our winning method, namely the one starting from a scale-free topology and producing a scale-free-like topology (SF2SFrand) can reduce training time without sacrificing neural network accuracy and also cutting memory requirements for the storage of the neural network.

Keywords Scale-free · Network science · Model reduction · Training · Feed forward neural networks · Deep learning

1 Introduction

The tremendous success of deep learning in various domains such as image understanding, speech recognition, and data analysis, in general, has made neural networks among the most successful machine learning methods. Feed-forward networks comprised of fully connected layers of neurons form the basis for many of these popular neural architectures [11]. The connections among neurons are weighted, and the task of any neural training process is to set the value of these weights appropriately to minimize the value of a selected performance (error) function. For fully connected neurons, the number of weighted connections is quadratic concerning the number of neurons. Thus, the computational complexity of a training algorithm such as the popular back-propagation for a deep architecture will be a significant challenge for adopting a particular neural architecture. Recent trends such as tiny deep learning [28] and the deployment of deep learning in modern IoT environments such as the modern battlefield [26] make the development of fast neural network training methods as significant as their inference accuracy and even more critical.

There are various directions in speeding up the training process of a multilayer neural network, and they will be presented in more detail in Section 2. In summary, past efforts focused: a) on developing faster approaches than the traditional steepest descent optimization algorithm, b) on developing variations of back-propagation, e.g., with adaptive/different learning rates, c) on the technique of dropout, d) on neural network topology pruning/sparsification, and e) on hardware-based solutions.

The research work is supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5631)

✉ Dimitrios Katsaros
dkatsar@e-ce.uth.gr

Evangelia Fragkou
efragkou@e-ce.uth.gr

Marianna Koultouki
mkoultouki@e-ce.uth.gr

[1] Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece

In this work, we develop methods for defining how neurons from adjacent layers are connected to each other, or to tell it in different words, we define methods for altering the topological structure of the neural network before, during, and after its training. Here, starting from the observation that biological neural networks are not fully connected but sparse, and they are scale-free or small-world [4], we investigate the usefulness of network science concepts, and in particular, the use- fulness of the theories concerning complex network growth models in speeding up a neural network training. Network science is the discipline that studies complex networks and their properties, and it has already proven its value in online social networks [3], wireless networking, and other fields, in being - apart from a "descriptive" discipline an arsenal for developing solutions for many problems. In particular, we examine the impact of scale-free and small-world topologies (instead of the traditional bipartite fully-connected ones) on the training speed and the accuracy in training data of a neural network.

1.1 Motivation and contributions

The emergence of fields such IoT networks and TinyML which necessitate the need of having a learning process appropriate for running in resource-starving devices e.g., sensors, have issued significant research questions – among others – on how to reduce the number of parameters (weights) of a neural network; such a reduced model implies a reduction of both the computational power needed to set these weights for instance via backpropagation, and also a reduction in the storage (memory) requirements [5] that this huge set of weights generates for the small/tiny device hosting the training process. Therefore, techniques falling into the generic family of model reduction and being termed as neural network sparsification [15] have been a very intense area of investigation. Indeed, topology sparsification has proved itself as a very promising technique for speeding up neural network training. The efficacy of this family of methods is supported by the fact that among the vast number of trainable weights that a neural topology has, only a moderate percentage of them differs significantly from zero; in other words the near-zero-weight links that connect neurons could be erased. Indeed some methods adopt such sparsification decisions after the training phase, in order to speed up the inference (operating) phase of the network [12].

This paper addresses the problem of neural model reductions from a novel perspective. We get inspiration from findings that actual biological neural networks are scale-free [4]. Thus, we explore network science tools to reduce the number of connections (weights), according to a specific rule. We take a principled approach in the sense that we seek to produce a particularly structured topology, e.g., a scale free topology starting from another structured topology, e.g., a small-world topology. The only prior work investigating such an approach is reported in [22]; however their starting point is a 'random' network according to the Erdos-Renyi notion [1]. This initiallizing method might end up being not efficient at all, because depending on the linkage density (controlled by a hard-to-set parameter) the initial network might be too dense (and almost fully connected) or too sparse (and unable to reach a scale-free topology). It might take too long to reach a structured (scale-free, or small-world) topology, starting from a purely random topology. Moreover, we depart from the common approach where pruning is done after training, and we perform pruning after each epoch; contrary to the intuition, this technique does not give bad results at all, but it is very competitive and even superior in many cases. In that context, this paper makes the following contributions:

- It puts forward a systematic effort to investigate the toolbox of network science to accelerate neural network training.
- It proposes algorithms to construct structured neural topologies starting from other structured neural topologies, all different from fully connected bipartite ones, to speed up the training phase of feed forward neural networks.
- It evaluates the performance of these algorithms, and confirms their rationale. It marks a winner algorithm and detects its features that makes it prevail.

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

2 Related work

The research on speeding up neural network training 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 is very slow. Therefore, methods based on second order Taylor series were investigated, such as Newton's method. Other algorithms that departed from the first order gradient concept, are those based on conjugate gradient, and the similar in spirit quasi-Newton method of Broyden-Fletcher-Goldfarb-Shanno (BFGS), along with its variations, e.g., L-BFGS [23]. Recently, fast optimizers have been proposed such as Adam, Adadelta, Nadam [9, 29].

Another family for accelerating neural training is that based on developing variable learning rates [17].

The topic of network architecture search [8, 30], which aims to design a neural architecture that achieves the best possible performance using limited computing resources in an automated way with minimal human intervention, bears similarity with our work and it is a very hot research area [14]. However, these methods may consume hundreds of GPU days or even more computing resources, have huge search spaces, too many hyperparameters, and difficult to achieve tricks.

The technique of dropout [32] constitutes the founding member of a new family, which randomly drops neurons during training. Similar in spirit, are the methods which compute only a subset of gradients during back propagation, e.g., meProp [33, 34] which prunes neurons based on how many times the back propagation updates a neuron.

Hardware-based accelerators comprise another familty and architectures such as FPGAs, multicore CPUs [38], TPUs [19] are increasingly used for neural training and inference.

A recent, very promising line of research [10, 31] suggests that only a subnetwork of the topology is responsible for carrying out accurately a particular task, and thus if we can detect which is this subnetwork and then train only this, pruning the rest of the network, then we can gain significant speedups in training. However, the complete understanding of tradeoffs involved in pruning is yet to be understood [20].

The family of algorithms related mostly to our work are those based on topology sparsification [15]. Methods have been proposed which prune connections between the neurons; for instance, the works [13], [24] and [6, 7, 21]. However, these linkage sparsification techniques do not aim at mimicking the topological structure of real neural networks, but are mainly based on eliminating close-to-zero weighted connections. The most closely related work to ours is that reported in [22], but they start from a completely unstructured topology basis, i.e., purely random network that might compromise performance. Contrary to work [35] referred to Convolutional Neural Networks (CNNs), we implement clear, close-to-zero weights pruning because of the nature of the network we use. MLPs perform simple operations among the connections of neurons for every layer, while CNNs use a different way of handling input features, by learning a specific section of them in every layer. So, in MLPs, if a weight value is very small that tends to be zero, it will not actually, contribute to the network weights update and consequently it does not affect the network performance but increases training time, so it can be deleted. On the other hand, removing arbitrarily weights on CNNs, even with negligible value, then we risk, losing crucial parts of inputs, given and hence reducing the accuracy of the model.

Overall, the methods developed in the present 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, and we take a first step in walking this avenue.

3 Background on network science concepts

Network science is the discipline that analyzes the properties and function of complex networks, such as technological, social, biological, physical and so on. Complex network analysis consists of algorithms and methodologies for studying and developing: centralities, communities, network growth and the respective models [1], diffusion processes [3], etc. Well-studied network models comprise random networks, regular lattices, small-world networks, and scale-free networks.

Definition 1 A random (or Erdos-Renyi) network is a network that consists of n nodes where each node pair is connected with probability p. The degree distribution of a random network follows a Poisson distribution.

Definition 2 A regular lattice is a network consisting of n nodes where each node has the same number and pattern of connections with every other node in the network. The degree distribution of a regular lattice is uniform (constant).

Definition 3 A small-world (or Watts-Strogatz) network that consists of n nodes interpolates between a regular lattice and a random network. In other words, it presents the regularity in neighborhood connectivity that a regular network has, but it also has some random connections towards random nodes of the topology. The degree distribution of a small-world network follows a Poisson-like bounded degree distribution.

Definition 4 A scale-free network is a network that consists of n nodes whose degree distribution follows a power-law; i.e., there are hub nodes (highly connected nodes) at any scale of observation. A particular type of scale-free network is generated by a process known as preferential attachment - known as Barabasi-Albert (BA) model [2] - according to which when a new node is added into the network it wires itself towards nodes which have already high degrees.

An example of these networks, namely regular, small-world, random and scale-free is depicted in Fig. 1.

4 The proposed techniques

This section describes five algorithms for creating topologies inspired by network science. They all share the common feature, namely they start from a structured topology and following and algorithmic procedure, they produce another structured topology. Before presenting the details and the pseudo-code for each of the methods developed here, we provide a short description of the randomized technique for rewiring the links among nodes which constitutes the basis of our algorithms.

4.1 Scale-free to scale-free with random rewiring (SF2SFrand)

In our first method, we start from a scale-free network and we end up by creating a new topology which is again scale-free based on the following principle:

– We create a sparse table, representing the connections between the nodes in each layer.
– Then, we remove the weights close to zero (sparsity) and then add as many links as we removed to the most powerful node. If a link that has to be reconnected, already exists, then no change takes place.

The main difference between SET and SF2SFrand lies in the construction of the initial network topology. SET creates the initial network connecting arbitrarily pairs of nodes. On the contrary, SF2SFrand initializes the network immediately as a scale-free topology; i.e., SF2SFrand connects each node of every layer to the next layer's most "powerful" node. Thus, SF2SFrand starts from a power-law (scale-free) topology. Even though this seems as a slight deviation, in practise it makes a great difference in performance, and it also needs intelligent, algorithmic solution to the definition and selection of a "powerful" node (described in the next paragraph). SF2SFrand's different initial topology reinforces weights with a strong value and in this case important information is kept and passed through layers in the training phase. Consequently, SF2SFrand achieves faster convergence due to well-organized and well-distributed information in the early neural network. Moreover, its accuracy increases abruptly in the early training iterations. Thus, the always-scale-free topology of SF2SFrand results in significantly less time to converge. (cf. Section 5.2). Finally, the purely random rewiring choices of SET implies that all attempted reconnections succeed, whereas the principled rewiring choices of SF2SFrand might result in that some attempted reconnections already exist because too many nodes are already connected to the powerful/hub node; thus we earn a further reduction in the number of weights.

In order to find the most powerful node, we calculate the link-attraction probability of each node, which is defined as the ratio of the incoming connections of this node to the total number of connections present in the graph. Every link is reconnected to the node with the largest such probability. We end up with a graph similar to scale-free, with the property that during the training it removes the links with weight close to zero and adds as many links as removed randomly after each epoch. The algorithm is illustrated in Algorithm 1.

```
1. Initialize a sparse table randomly;
2. Remove links, whose weights are close to zero;
3. while (each node i of every layer) do
    4. calculate the maximum degree probability as follows:
    5. $p_i = \sum(incoming\_links_i)/\sum(links\_in\_network)$;
end
6. Reconnect the nodes,whose link was removed,with the node that has the maximum p;
7. The weight of the new connection is given, randomly;
8. if (this link exists) then
    9. nothing is done;
end
10. Weights Evolution via BackPropagation();
11. while (each epoch) and during BackPropagation() do
    12. remove links, whose weights are close to zero;
    13. reconnect the nodes, whose link was removed, randomly;
    14. the weight of the new connection is given, randomly;
end
```

Algorithm 1 Pseudocode of Scale-Free to Scale-Free algorithm with random rewiring (SF2SFrand).

4.2 Scale-free to scale-free using preferential attachment (SF2SFba)

In this algorithm, a scale-free topology is constructed, which means that links are connected to the nodes such that the degree distribution follows a power-law. In other words, links are added to the node according to their current degree [2] implementing a Barabasi-Albert attachment policy.

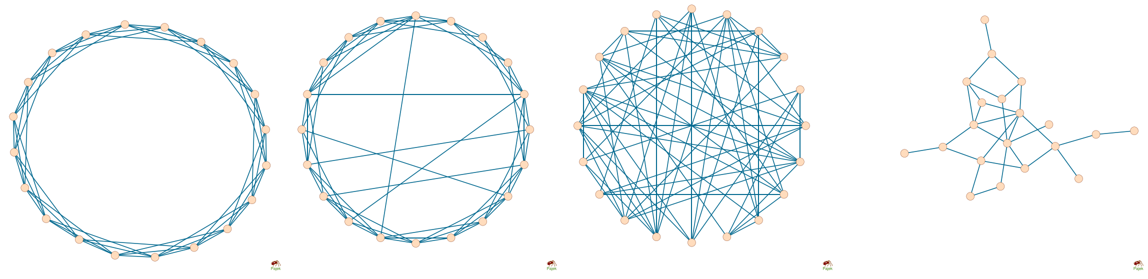


Fig. 1 **(a)** A 20-node regular network (with 3 neighbors at each side of every vertex), **(b)** a 20-node small-world network (with initially 3 neighbors at each side of every vertex, and with 0.075 rewiring probability), **(c)** a 20-node random network (with average node degree 6) and **(d)** a 20-node scale-free network. Graphs were generated with Pajek

We start by creating a scale-free network, using the algorithm described in Section 4.1. Then, 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. The algorithm is depicted in Algorithm 2.

```
1. initialize a sparse table randomly;
2. remove links, whose weights are close to zero;
3.
4. while (each node i of every layer) do
    5. calculate the maximum degree probability as follows:
    6. $p_i = \sum(incoming\_links_i) / \sum(links\_in\_network)$;
end
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. if (this link exists) then
    12. nothing is done;
end
13.
14.Weights Evolution via BackPropagation();
15.while (each epoch) and during BackPropagation() do
    16. while (each layer) do
        17. do steps 2 to 12;
    end
end
```

Algorithm 2 Pseudocode of Scale-Free to Scale-Free algorithm, using preferential attachment (SF2SFba).

4.3 Scale-free to scale-free (5 strongest nodes) (SF2SF5)

In this particular algorithm, we start with a scale-free implementation, as described in Section 4.1 and end up in a similar type of network, using an alternative version of scale-free technique. 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 the original method, if a link, from one node, in a layer, to another, which has to be reconnected, already exists, then changes do not occur. 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 potent node and so on, untill the fifth strongest node. The weights are randomly given in the original version and in our alternative one.

We developed 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 those that have to be reconnected, in the network. The algorithm is illustrated in Algorithm 3.

4.4 Scale-free to small-world (SF2SW)

Here, we start again from 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 critical impact, we calculate the degree probability of every node and then reconnect these nodes (in every layer), whose link was deleted, to the most potent node of the next layer. After the training part of the algorithm we proposed, a small-world network is created.

```
1. initialize a sparse table randomly;
2. remove links, whose weights are close to zero;
3.
4. while (each node i of every layer) do
    5. calculate the maximum degree probability as follows:
    6. $p_i = \sum(incoming\_links_i) / \sum(links\_in\_network)$;
end
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. if (this link exists) then
    12. nothing is done;
end
13.
14.Weights Evolution via BackPropagation();
15. while (each epoch) and during BackPropagation();
do
    16. while (each layer) do
        17. while (each node) do
            18. do steps 4 to 7;
            19. reconnect all links removed,as follows:
            20. while (each j in 5 strongest nodes of next layer) do
                21. if (the link in j node does not exist) then
                    22. connect node to j;
                    23. break;
                end
            end
        end
    end
end
```

Algorithm 3 Pseudocode of Scale-free to Scale-free (5 strongest nodes) (SF2SF5) algorithm.

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). Then, 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 the 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 does not exist, else, we try to find another random node to connect to). In this case, 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 training process; we create a network with a more arbitrary structure. The algorithm is illustrated in Algorithm 4.

```
1. initialize a sparse table randomly;
2. remove links, whose weights are close to zero;
3.
4. while (each node i of every layer) do
    5. calculate the maximum degree probability as follows:
    6. $p_i = \sum(incoming\_links_i) / \sum(links\_in\_network)$;
end
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. if (this link exists) then
    12. nothing is done;
end
13.
14.Weights Evolution via BackPropagation();
15. while (each epoch)and during BackPropagation()
do
    16. define a threshold probability $P$;
    17. while (each layer) do
        18. while (each node n in this layer) do
            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. while (each N) do
                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;
            end
        end
    end
end
```

Algorithm 4 Scale-Free to Small-World (SF2SW) pseudocode.

4.5 Small-world to small-world (SW2SW)

In the last algorithm, we implement a network based on small-world technique, and the same type of network is constructed through the training process. It is interesting to see how the transition from a less randomly constructed network (small-world) to another affects the network performance. These small-world networks are created in the same way described in Algorithm 4. We show the pseudocode of this new method in Algorithm 5.

```
1. initialize a sparse table randomly;
2. define a threshold probability $P$;
3. while (for each layer) do
    4. while (each node in this layer) do
        5. find links, whose weight is non-zero;
        6. give those links a random probability $P_{node}$;
        7. find numbers N of nodes, so as: $P_{node} < P$;
        8. while (each N) do
            9. select a node m in next layer randomly,
            so that its link weight is zero;
            10. rewire current node to node m;
        end
    end
end
11.
12.Weights Evolution via BackPropagation();
13. while (each epoch and during BackPropagation())
do
    14. do steps 2 to 10;
end
```

Algorithm 5 Small-World to Small-World (SW2SW) pseudocode.

5 Experimental evaluation

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

5.1 Evaluation settings

In this work, we deal with a classification task using data regarding the disease of cancer and fashion.

Competitors In our experiments, we used as competitors the algorithms proposed in the current work and presented in Sections 4.1–4.5. We use the following keys for the proposed algorithms: *SF2SFrand* for Algorithm 1, *SF2SFba* for Algorithm 2, *SF2SF5* for Algorithm 3, *SF2SW* for Algorithm 4 with two versions based on different probabilities, i.e., *SF2SW(p=0.02)* and *SF2SW(p=0.075)*, *SW2SW* for Algorithm 5 with two versions based on different probabilities, namely *SW2SW(p=0.02)* and *SW2SW(p=0.075)*. The competitors include the SET algorithm [22], and also the fully-connected (FC) neural network without any pruning. Moreover, we implemented a recent algorithm inspired by the concept, proposed in [6] and a set of variations implementing the logic of the algorithm reported in [35]. We call the former algorithm as the ζ *- parameterized - (SF2SFrand)*, which is a variant of SF2SFrand and it differs in the values of ζ parameter, which is responsible for the number of the close-to-zero weight links, being erased in every epoch. In simple SF2SFrand, the value of ζ is stable during the training procedure while in the last implementation this value is tuned and specifically is reduced in every epoch, exponentially, so as to observe if the performance of the network is better, if different and gradually, smaller values of ζ are used. The set of the latter algorithms, called the *AVG-weighted algorithms* produce a new variant of each one of our algorithms. Specifically, they are modified only in the part which is responsible for assigning weight values to the new connections; i.e., instead of placing a random value as a connection weight, we aggregate the values of all the links, being pruned, in this layer, for every epoch and then we give the average value of this sum, as the weight of a new connection. For every new incoming link, we assign this value, being modified about 2%, each time.

Datasets In order to evaluate our algorithms, we use the classification accuracy as a measure, which is the percentage evaluation between the predicted and target value. Specifically, we present the evolution of accuracy, in every epoch, during the whole training process, until the model converges. Furthermore, we show the time, needed for a model to generalize the given data.

In order to test the algorithms, we used datasets that have a hundred instances and a few thousand features and they are encoded with one-hot encoding, as described in Table 1.

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

Regarding the technical details, we used a laptop computer with 6GB of RAM and an Intel Core i7 processor regarding all datasets except Fashion - Mnist, which was tested on a desktop computer with 16 GB of RAM and an Intel Xeon W 2123 processor, clocked at 3.60 GHz.

Lung and lung_discrete These data were used by Hong and Young to illustrate the power of the optimal discriminant

Table 1 Datasets characteristics

Filename(.mat)	Instances	Features	Classes
lung	203	3312	5
lung_discrete	73	325	7
TOX_171	171	5748	4
CLL_SUB_111	111	11340	3
COIL20	1440	1024	20
Fashion-Mnist	70000	28x28	10

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 three types of pathological lung cancers. The authors give no information on the individual variables nor on where the data was originally used [16].

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 [18].

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 1 [18].

COIL20 Columbia Object Image Library (COIL-20) is a database of gray-scale images of 20 objects. The objects were placed on a motorized turntable against a black background. The turntable was rotated through 360 degrees to vary object pose with respect to a fixed camera. Images of the objects were taken at pose intervals of 5 degrees. This corresponds to 72 images per object. The database has two sets of images. The first set contains 720 unprocessed images of 10 objects. The second contains 1,440 size normalized images of 20 objects. COIL-20 is available online via ftp [25].

Fashion-Mnist Fashion-MNIST is a new dataset, consisting of 70,000 samples, that represent fashion items. Each one of them is a 28 $\times$ 28 grayscale image, belonging to one of 10 different categories. There are about 7,000 samples per category. The dataset is separated from the training set and the testing set, which contains 60,000 and 10,000 images, respectively. Fashion-MNIST is intended to replace the original MNIST dataset for benchmarking machine learning algorithms, taking into account that it has the same structure as MNIST does [36].

Furthermore, **the first 5 datasets, mentioned in** Table 1, are split in two parts, the training and the testing set. The $\frac{2}{3}$ of the total size of the dataset is used in order for the neural network to be trained, and the $\frac{1}{3}$, remaining, is used for testing its ability to learn the training set. Regarding the last dataset, Fashion-Mnist is already separated in training and testing parts, with 60,000 and 10,000 samples, respectively.

The method, used for preprocessing features, in order for them to be converted into a form that can be more "understandable" to our deep learning algorithms, is one-hot encoding, which creates new (binary) columns, indicating the presence of each possible value from the original data.

An MLP (Multilayer Perceptron) neural network model is used in all cases. More specifically, it is about a neural network with an input level, three hidden layers, and one output level. Each hidden layer has 1,000 neurons, and the number of neurons in the input and output levels depends on the dataset features. The application of our algorithms to other feed-forward networks, such as convolutional neural networks (CNN), is possible. So, It is necessary to deal with their particularities and introduce some more algorithmic techniques, which we are currently developing in order for our algorithms to be adapted to the function of the convolutional-pooling layers instead of simply introducing the existing techniques into the final fully-connected layers of a CNN, as it was done by the SET algorithm [22]. Moreover, inspiration from the ideas developed in [37] could also assist in this goal.

Optimizers and activation functions In our experiments, we use stochastic gradient descent with momentum with parameter γ equal to 0.9. For the learning rate α and batch size parameter bs, we used the values of 0.01 and 2, respectively.

Table 2 List of hyperparameters, being used

List of hyperparameters	
Number of layers	**1 input layer (with #neurons,equal to the features of dataset)** 3 **hidden layers** **1 output layer (with #neurons,equal to the classes of dataset)**
Number of neurons per layer	1000
Activation functions	ReLU
	FReLU
	Sigmoid (in last layer)
Weights regularization techniques	L1, L2
Weight_decay ($wdec$)	0.0000001 for L1
	0.0002 for L2
Batch size (bs)	2
Learning_rate (α)	0.001
Momentum (γ)	0.9
Percentage of connections pruned/restored (ζ)	30%

As activation functions we used the popular ReLU and FReLU functions for the hidden layers, and the sigmoid function for the last level. The activation functions we used are described below. The Rectified Linear Unit (ReLU) is defined as follows : $ReLU(x) = (x > 0)?x : 0$, and the Flexible Rectified Linear Unit (FReLU) is defined in [27]

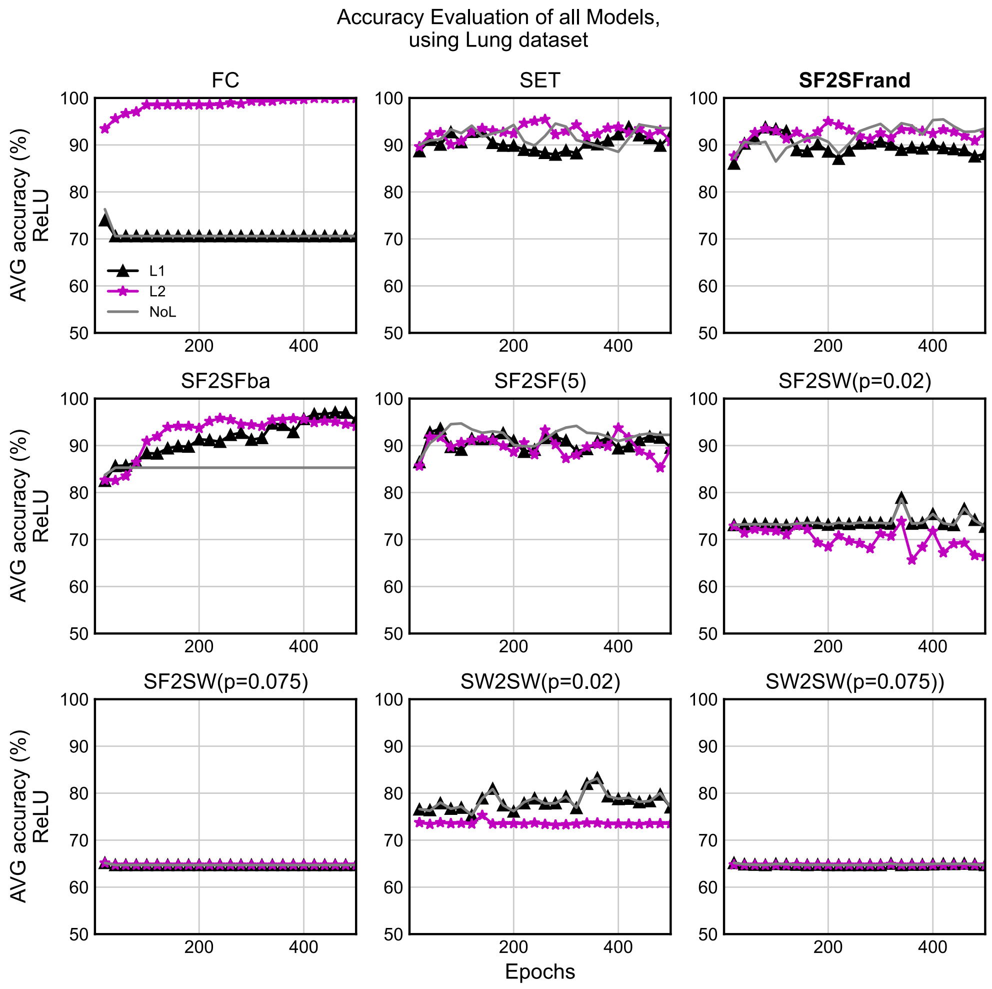


Fig. 2 Lung dataset overall results, using ReLU activation function

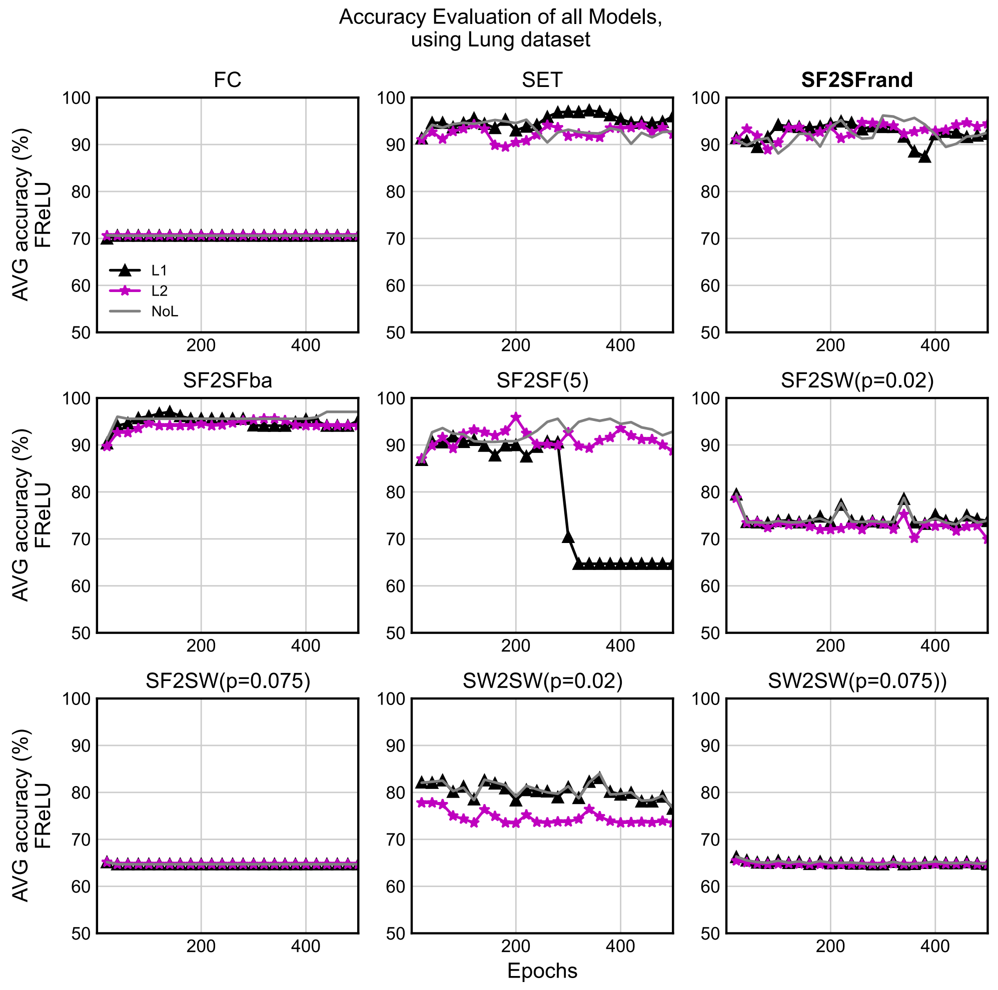
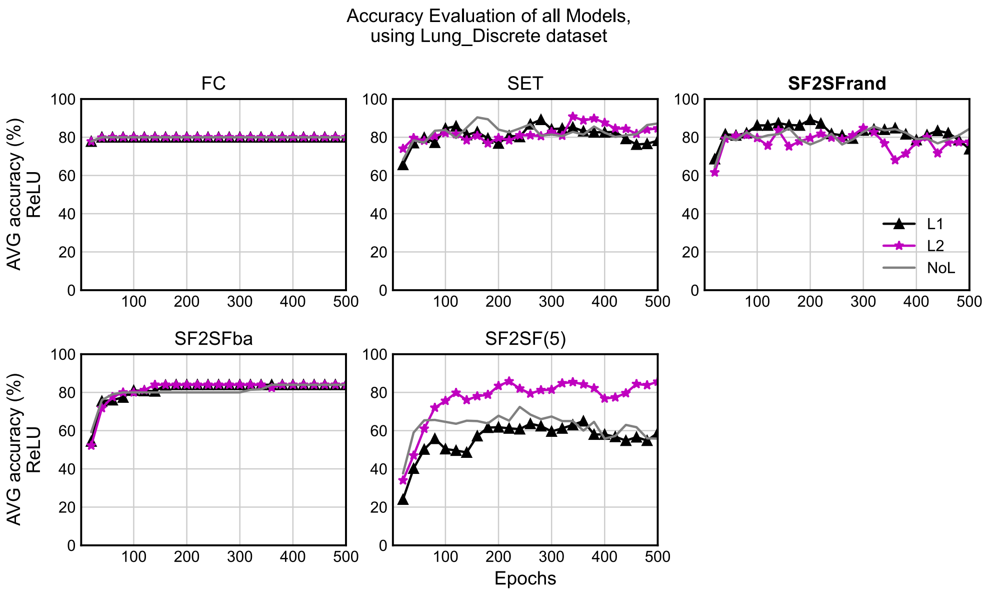


Fig. 3 Lung dataset overall results, using FReLU activation function

Fig. 4 Lung Discrete dataset overall results, using ReLU activation function



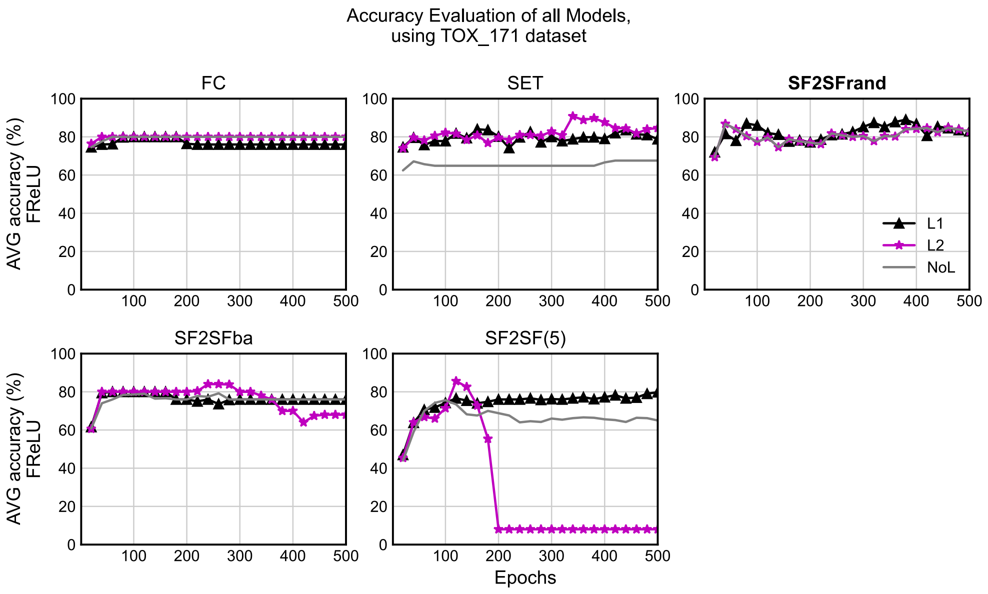


Fig. 5 Lung Discrete dataset overall results, using FReLU activation function

as follows: $FReLU(x) = ReLU(x) + b$ with b being a learnable parameter.

All algorithms functioned repetitively until convergence; here, for visual comparison purposes, we show the performance after the first 500 epochs when we practically have convergence, except for some very few cases where converge is achieved after 50 epochs. We also show the accuracy calculated at each time according to the mean squares error method. Each training attempt was repeated five times, and the performance was averaged.

Finally, in cases where we implement regularization techniques, the weight decay parameter $(wdec)$ was set to 0.0002 for L2 regularization, and 0.0000001 for L1 regularization method. Moreover, the number of connections being modified depends on the value of ζ parameter, which is set to 0.3 in our experiments. In other words, the $\zeta = 30\%$ of the real connections of the network are pruned and restored in every epoch. **All hyperparameters used are illustrated in** Table 2. We use prediction accuracy as the performance measure.

In the following subsections, we show for each dataset and each competitor the "evolution" of the accuracy achieved until the last epoch; at the end of these sets of plots, we show aggregate results concerning the average accuracy

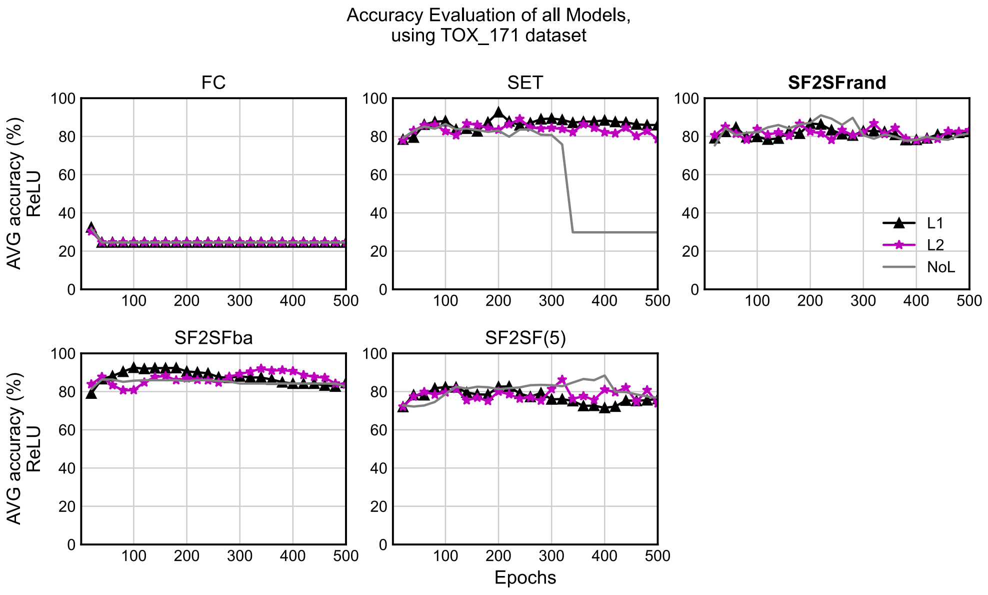
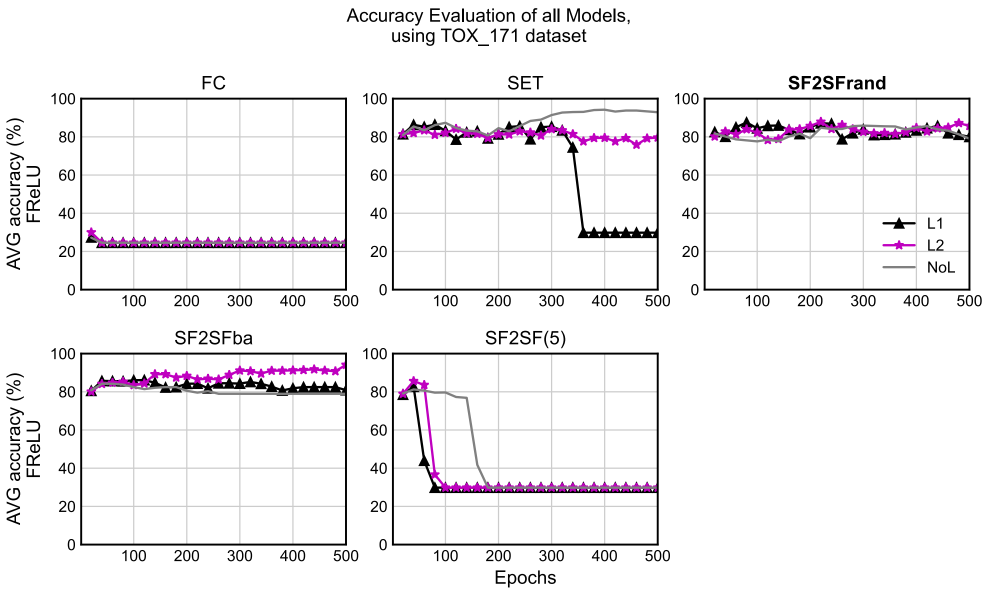


Fig. 6 TOX_171 dataset overall results, using ReLU activation function

Fig. 7 TOX_171 dataset overall results, using FReLU activation function



achieved during the last epochs, and also the average total training time required. However, some methods did not converge in "reasonable" time while processing some of the datasets, so these measurements are excluded from the presented results.

5.2 Evaluation results

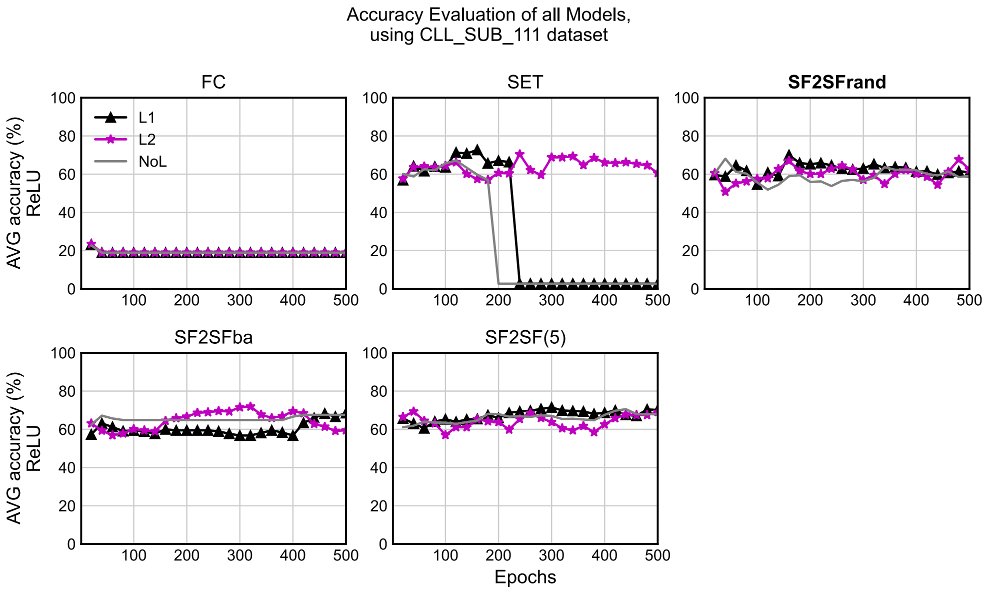
In all figures presented in the sequel, the x-axis represents the epochs; each one of them is a single step in training a neural network. The y-axis describes the percentage evaluation between the predicted and target value (i.e., accuracy).

In the subsequent figures, we present the results of our methods in comparison with the accuracy of both the Fully-Connected (FC) MLP network and the results of the SET algorithm in all datasets mentioned.

Figures 2 and 3 display all algorithms' accuracy, taking into account the Lung.mat dataset. Specifically, the accuracy that all algorithms achieve is depicted with respect to the epochs.

In the top Fig. 2, we see the results that ReLU activation produces for Lung.mat dataset, and in the bottom plot, the results produced by FReLU function. When we apply the SF2SFrand algorithm (which starts from a randomly constructed network and ends up in a scale-free-like one),

Fig. 8 CLL_SUB_111 dataset overall results, using ReLU activation function



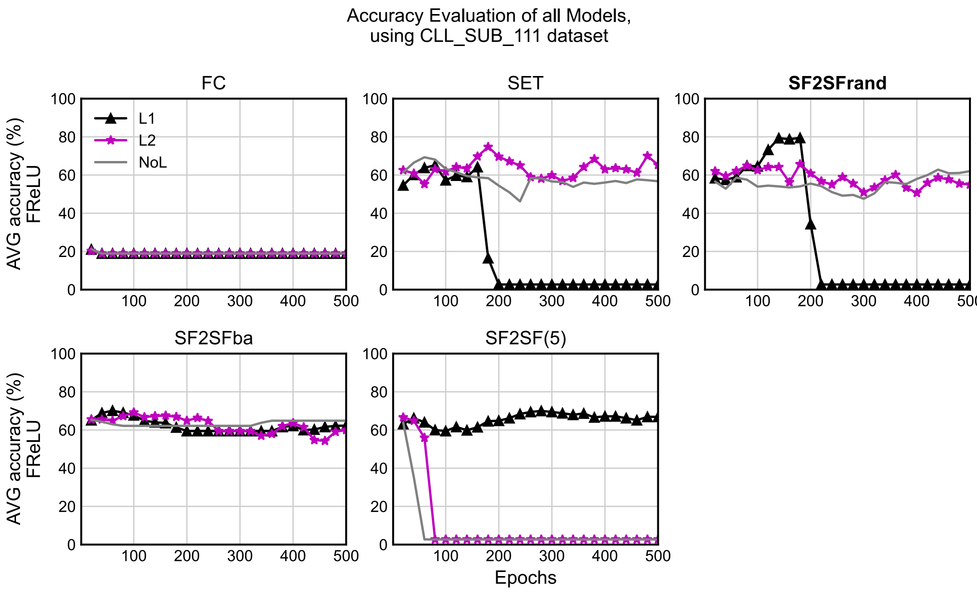


Fig. 9 CLL_SUB_111 dataset overall results, using FReLU activation function

we see that the accuracy varies from 70% to 99%, and we conclude that the algorithm has a better performance when L2 regularization is used (approximately 93% accuracy, when using FReLU).

So, L2 (L2 regularization makes the loss function smoother) manages to decrease the noise in the training data so that the estimated coefficients (weights) can generalize well to the future data. According to the time, the best performance is achieved in 10 minutes when SF2SFrand is used, which means this kind of modification that this algorithm does in topology, positively affects the network. Similar performances are shown in Figs. 4, 5, 6, 7, 8, 9, 10 and 11 during the execution of the algorithm, using the other datasets which are mentioned in Table 1.

In Figs. 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11, we illustrate the results for the algorithm that starts from a scale-free network and, through the procedure of training using back-propagation, ends up constructing a scale-free network again (SF2SFba). This means that the network is always modified following a power law, and hence the randomness is decreased. Using SF2SFba, in turn, implies better communication between the nodes. According to the plot, this variant of the algorithm has 95.6% accuracy using FReLU and without any regularization. So, taking into consideration that FReLU has faster convergence than ReLU [27], 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, the algorithm takes 29 minutes (with 92% accuracy) to train the network, using ReLU (recall that this function is computationally efficient since it just outputs zero for negative inputs) and 35 minutes while implementing the FReLU, which is still better in both time and accuracy than the competitor.

Furthermore, the exact figures show the accuracy of a variant of scale-free to a scale-free algorithm, described in Section 4.3. The 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 the network performance positively, except in 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 error in weight update (appears nan values). The algorithm takes approximately the same time to train the network as scale-free to scale-free version does.

Our following technique was the one that started from a scale-free and created a small-world network (SF2SW). Our results (when using p=0.02, aw rewiring probability) show that the accuracy is decreased when a network, following a power-law degree distribution, transits to a more randomly linked topology.

Specifically, in Figures 2, 3, 10 and 11 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 also has a much larger execution time than the rest, owing to its great computational complexity, as Algorithm 4 implies.

We run the same algorithm with a more significant probability (p=0.075) and show the results in the exact figures. While the probability becomes larger than the previous one, the average length becomes smaller, which

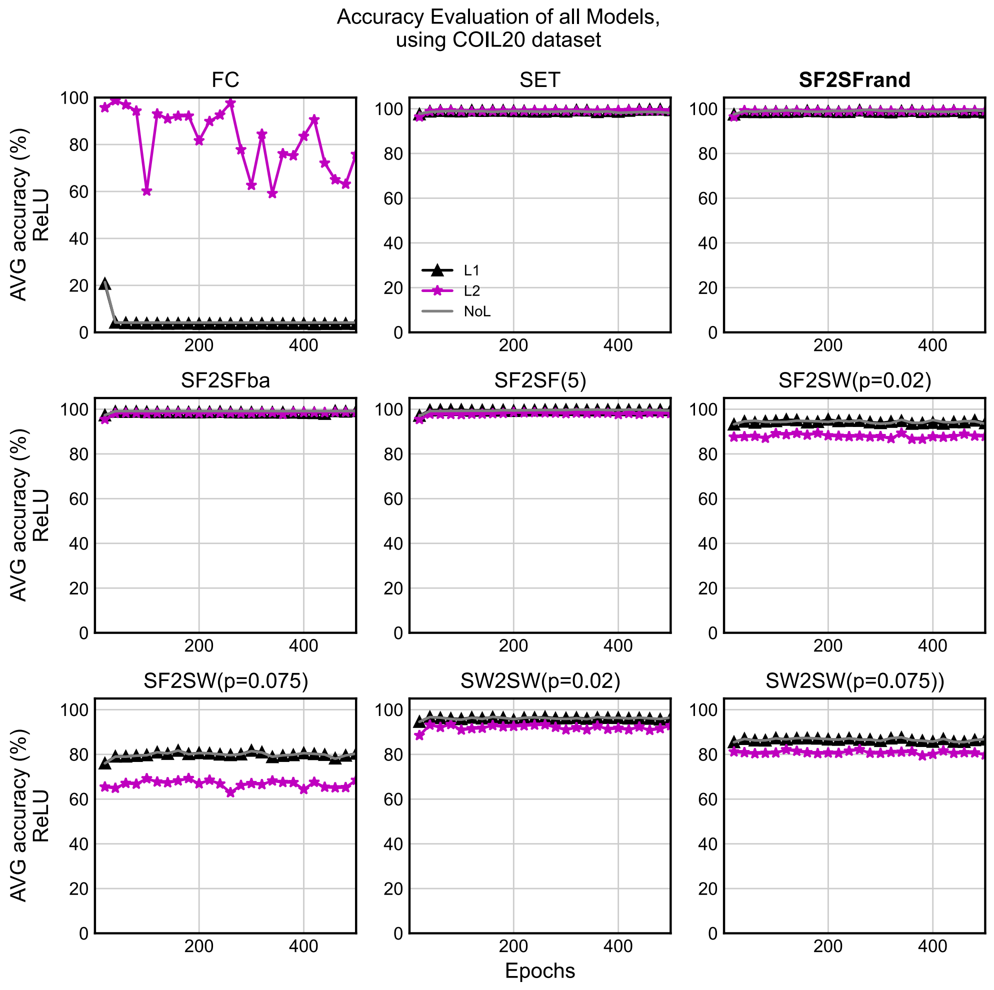


Fig. 10 COIL20 dataset overall results, using ReLU activation function

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 the same random graph as in Algorithm 4. In particular, a node can 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.

In this last case, we study the performance of the neural network system in terms of using only the small-world method and also depict the results in Figs. 2, 3, 10 and 11. Using FReLU activation, the 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), meaning the reconstructed network is not random enough. Hence the accuracy is stabilized in high levels of values. Therefore, the less a topology is random, the more efficient the network becomes, in learning and generalizing the data.

Then we use a much larger probability, 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. Those affect the network and so as the accuracy, which is, slow enough (65%). The graphs show that L1 regularized curve has many fluctuations while FReLU function is used. This is because FReLU provides more capacity than ReLU, which leads the model to not generalize well from its training data to unseen data. Moreover, the time needed for the model to be trained while using the Scale-Free-to-Small-World or the Small-World-to-Small-World algorithm is vast enough and the accuracy is moderate regarding the other algorithms' performance, so we test these algorithms only to these two datasets

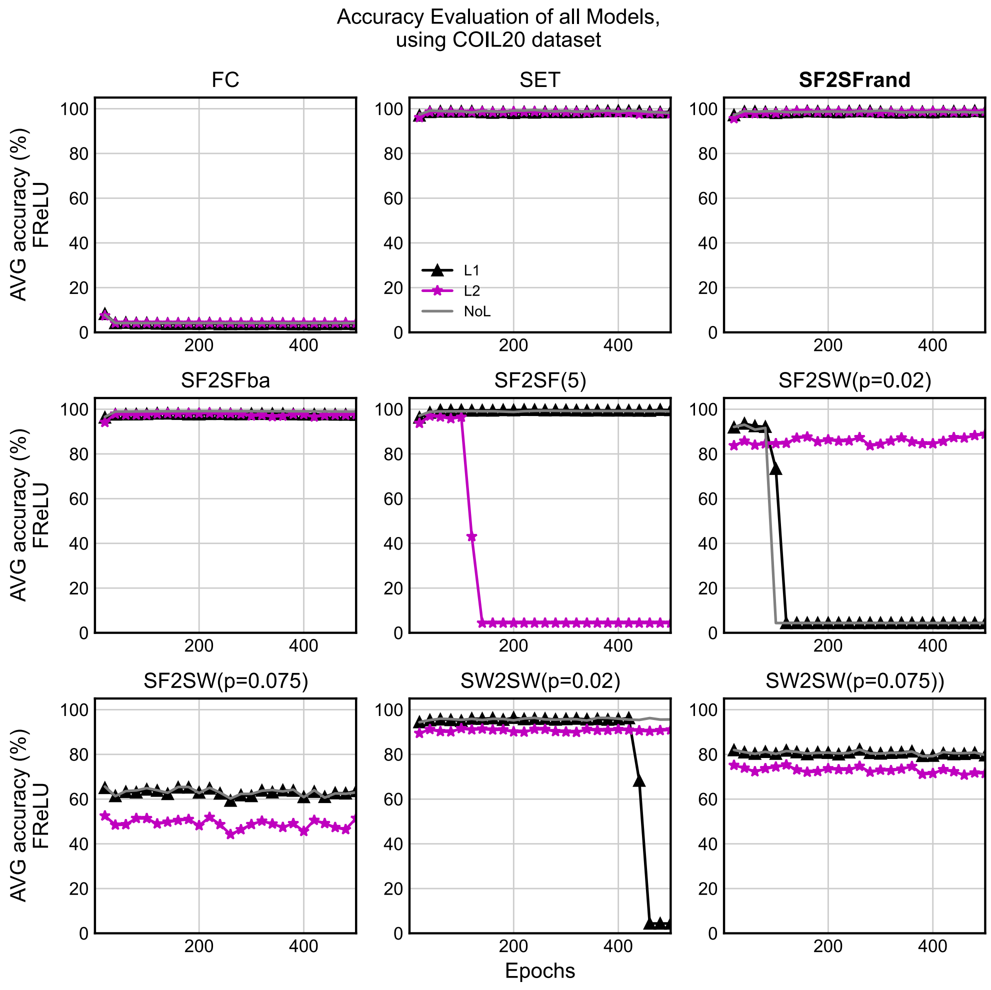


Fig. 11 COIL20 dataset overall results, using FReLU activation function

(Lung.mat and COIL20.mat - we chose specifically these, due to the different classification task they do.)

It is worth highlighting that we compare our sparse model results with the performance of the fully-connected, unpruned neural network. As we can see in the above figures, the accuracy of the fully-connected model can be high enough (up to 95% in some cases) mainly when we have the combination of ReLU and L1 regularization, in

Table 3 Memory footprint in MegaBytes(MB), between the Dense Network (MLP), the Sparse Network using SET algorithm and our proposed algorithms, which are SF2SFrand, SF2SFba abd SF2SF(5), SF2SW and SW2SW

Memory Footprint				
Algorithm	Kind of Network	Hidden Layer Architecture	Size (MB)	Percentage reduction
MLP (FC)	Fully-Connected	1000 - 1000 - 1000	77.68	—
SET	Sparse	1000 - 1000 - 1000	19.18	75.28 %
SF2SFrand	Sparse	1000 - 1000 - 1000	18.39	76.31 %
SF2SFba	Sparse	1000 - 1000 - 1000	20.68	73.36 %
SF2SF(5)	Sparse	1000 - 1000 - 1000	20.18	74.00 %
SF2SW	Sparse	1000 - 1000 - 1000	19.84	74.44 %
SW2SW	Sparse	1000 - 1000 - 1000	21.30	72.55 %

Table 4 Measures of MLP (FC)

Measures - MLP (FC)					
Datasets- Activ.Fun.+Regul	Lung *Acc/Time*	Lung_Discrete *Acc/Time*	TOX_171 *Acc/Time*	CLL_SUB_111 *Acc/Time*	COIL20 *Acc/Time*
ReLU + NoL	70.58% /1h 15 min	80% / 5 min	24.56% /75 min	19.07% /1h 5 min	4.1% /2h 8 min
ReLU + L1	70.58% /2h 22 min	80 % / 9 min	24.56% /1h 26 min	18.91% /1h 35 min	3.5% /9h 2 min
ReLU + L2	99.52% /1h 23 min	80 % / 7 min	24.56% /1h 21 min	18.91% /1h 31 min	62.5% /6h 3 min
FReLU + NoL	70.58% /1h 28 min	80 % /8 min	24.56% /1h 03 min	18.81% /1h 9 min	4.1% /2h 56 min
FReLU + L1	70.58% /2h	76 % / 11 min	24.56% /1h 37 min	18.91% /1h 39 min	3.5% /8h 15 min
FReLU + L2	70.58% /1h 26 min	80 % / 10 min	24.56% /1h 26 min	18.91% /1h 34 min	4.1% /3h 36 min

Table 5 Measures of SET

Measures - SET					
Datasets- Activ.Fun.+Regul	Lung *Acc/Time*	Lung_Discrete *Acc/Time*	TOX_171 *Acc/Time*	CLL_SUB_111 *Acc/Time*	COIL20 *Acc/Time*
ReLU + NoL	92.02% /35 min	82.59% / 11 min	38.9% /23 min	2.3% /45 min	98.75% /47 min
ReLU + L1	90.53% /47 min	80.90% /13 min	86.51% /35 min	3.3% /55 min	99.37% /1h 28 min
ReLU + L2	92.71% /38 min	82.10%/ 11 min	83.69% /27 min	65.84% /45 min	99.16% /54 min
FReLU + NoL	93.02% /45 min	65.50% /13 min	88.43% /32 min	58.15% /56 min	98.12% /1h 25 min
FReLU + L1	95.04% /53 min	79.62% /12 min	38.2% /36 min	2.1% /58 min	98.75% /2h
FReLU + L2	92.38% /49 min	82.11% /11 min	80.90% /33 min	63.54% /57 min	98.12% /1h 30 min

Table 6 Measures of SF2SFrand

Measures - SF2SFrand					
Datasets- Activ.Fun.+Regul	Lung *Acc/Time*	Lung_Discrete *Acc/Time*	TOX_171 *Acc/Time*	CLL_SUB_111 *Acc/Time*	COIL20 *Acc/Time*
ReLU + NoL	91.8% /10 min	80% /6 min	82.5% /10 min	58.9% /15 min	98.95% /36 min
ReLU + L1	89.6% /15 min	82.4% /8 min	81.4% /18 min	62.4% /19 min	97.91% /1h 8 min
ReLU + L2	92.3% /11	77.6% /7 min	81.6% /12 min	60% /14	98.95% /44 min
FReLU + NoL	92.1% /13 min	80.3% /6	82.3% /16 min	55.4% /18 min	98.75%/1h 14 min
FReLU + L1	92.4% /21	82.5% /7 min	83.3%/22 min	2.7% /17 min	99.16%/1h 42
FReLU + L2	92.9% /16 min	80.3% /6 min	83.3% /14	58.2% /15 min	98.54%/1h 28 min

Table 7 Measures of SF2SFba

Measures - SF2SFba					
Datasets- Activ.Fun.+Regul	Lung *Acc/Time*	Lung_Discrete *Acc/Time*	TOX_171 *Acc/Time*	CLL_SUB_111 *Acc/Time*	COIL20 *Acc/Time*
ReLU + NoL	85.2% /29	81% /15 min	84.8% /34 min	65.5% /1h	99.16% /53 min
ReLU + L1	91.5% /32 min	81.4% /18 min	87.5% /39	64.5% /1h 5 min	99.16% /1h 22 min
ReLU + L2	92.8% /30 min	82.4% /15 min	86.8% /37 min	60.4% /1h 7 min	98.75% /1h 18 min
FReLU + NoL	95.6% /32 min	78.9% /18 min	80.4% /38 min	63.3% /1h	98.95% /1h 29 min
FReLU + L1	94.9% /41	79.3% /20 min	83.5%/1h 19 min	62.4% /1h 5	97.5% /1h 58 min
FReLU + L2	94.1% /39	76% /18 min	95% /42 min	62.6% /1h 12 min	97.29% /1h 51

Table 8 Measures of SF2SF-5

Measures - SF2SF(5)					
Datasets- Activ.Fun.+Regul	Lung $Acc/Time$	Lung_Discrete $Acc/Time$	TOX_171 $Acc/Time$	CLL_SUB_111 $Acc/Time$	COIL20 $Acc/Time$
ReLU + NoL	92% /28 min	62.4% /13 min	81% /40 min	65.92% /1h 15 min	99.42% /41 min
ReLU + L1	90.60% /37 min	59.7% /13 min	77.2% /50 min	70.27% /1h 20 min	99.49% /45 min
ReLU + L2	89.84% /30 min	90.6% /14 min	77.2% /43 min	70.27% /1h 13 min	97.98% /45 min
FReLU + NoL	92.8% /35 min	70.1% /13 min	30.4% /39 min	9.8% /55 min	99.79% /1h 13 min
FReLU + L1	64.70% /37 min	80.3% /14 min	30.4% /34 min	66.7% /1h 26 min	99.79% /1h 25 min
FReLU + L2	91.2% /37 min	10.5% /11 min	30.4% /33 min	9.8% /57 min	4.3% /1h 7 min

Table 9 Measures of SF2SW

Measures - SF2SW					
p = 0.02			p = 0.075		
Datasets- Activ.Fun.+Regul.	Lung $Acc/Time$	COIL20 $Acc/Time$	Datasets- Activ.Fun.+Regul.	Lung $Acc/Time$	COIL20 $Acc/Time$
ReLU + NoL	74.7% /4h 13 min	94.36% /2h	ReLU + NoL	64.7% /6h 3 min	79.85% /4h 3 min
ReLU + L1	74.8% /4h 17 min	94.35% /2h 48 min	ReLU + L1	64.7% /6h 20 min	79.85% /4h 30 min
ReLU + L2	73.52% /4h 14 min	89.37% /2h 13 min	ReLU + L2	64.7% /6h 15 min	66.79% /4h 8 min
FReLU + NoL	74.52% /4h 20	04.37% /2h 19 min	FReLU + NoL	64.7% /6h 52 min	63.26% /4h 37 min
FReLU + L1	74.52% /4h 17 min	04.37% /3h 47 min	FReLU + L1	64.7% /6h 20 min	62.99% /5h 12 min
FReLU + L2	74.52% /4h 19 min	85.83% /2h 46 min	FReLU + L2	64.7% /6h 54 min	52.29% /4h 55 min

Table 10 Measures of SW2SW

Measures - SW2SW					
p = 0.02			p = 0.075		
Datasets- Activ.Fun.+Regul.	Lung $Acc/Time$	COIL20 $Acc/Time$	Datasets- Activ.Fun.+Regul.	Lung $Acc/Time$	COIL20 $Acc/Time$
ReLU + NoL	80% /4h 9 min	96.14% /2h 7 min	ReLU + NoL	64.8% /6h 3 min	86.58% /3h 28 min
ReLU + L1	80% /4h 12 min	96.15% /2h 44 min	ReLU + L1	64.8% /6h 55 min	86.57% /4h 4 min
ReLU + L2	78% /4h 8 min	95.41% /2h 7 min	ReLU + L2	64.7% /6h 6 min	80.81% /3h 35 min
FReLU + NoL	81% /4h 13 min	95.62% /2h 41 min	FReLU + NoL	65% /6h 5 min	80.67% /4h 4 min
FReLU + L1	81% /5h	04.37% /3h 14 min	FReLU + L1	65% /5h 50 min	80.63% /4h 40 min
FReLU + L2	78% /4h 15 min	90.62% /2h 52 min	FReLU + L2	65% /6h 11 min	73.00% /4h 10 min

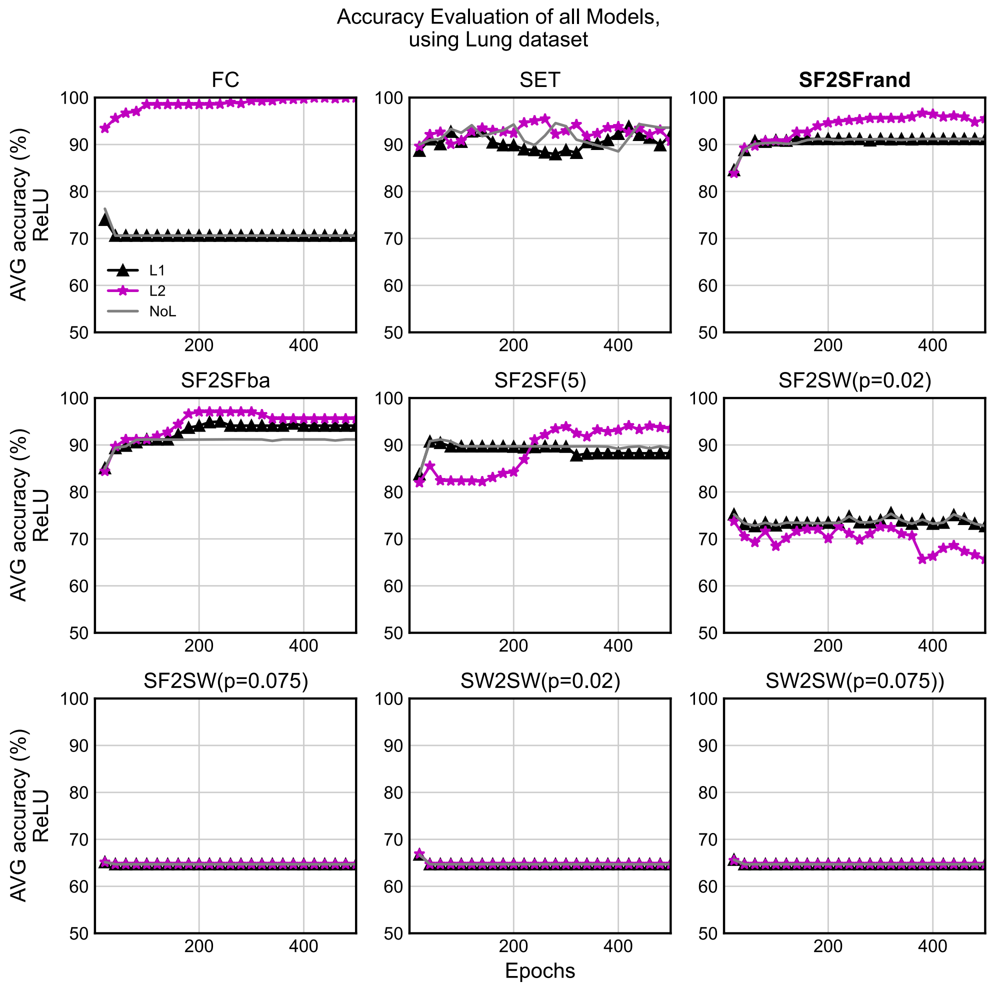


Fig. 12 Results of MLP(FC) and AVG-Weighted algorithms on Lung dataset, using ReLU activation function

contradiction to the cases, we use L1 or no regularization technique and FReLU activation function, in which the model achieves about to 20% accuracy or lower. However the main problems of a fully-connected model are its efficacy, comparatively with the time needed to be trained and its memory requirements, especially when we have devices with memory and storage constraints. In Table 3, we see the memory footprint of all the algorithms, mentioned and in Table 15 we see the overall results of them. In our approaches and specifically, when we implement a scale-free network topology, we reduce the network's training time up to 93% and the memory requirements for the models to be deployed up to 76% in comparison with the unmodified one. At the same time, we achieve better results regarding the accuracy in most cases and almost in all datasets used.

Furthermore, we achieved up to 76% reduction in model size, using the topology mentioned above (three hidden layers of one thousand neurons each of them) and the SF2SFrand algorithm, due to the redundant weights, removed, making our sparse implementations more suitable to run on devices with storage limitations, minimizing the computational cost. It is also of great importance to mention that one of our algorithms -the SF2SFrand algorithm- needs about 1% less storage than SET algorithm, while it has a better efficacy (it is much faster than SET while preserving approximately the same accuracy as SET)as it is shown in Table 6, which is the only former work, close to the concept of ours, presented in the literature Tables 4, 5, 7, 8, 9 and 10.

Hence, there is no reason to show the exact training time results of the unpruned model (due to the significant deviation, regarding our methods).

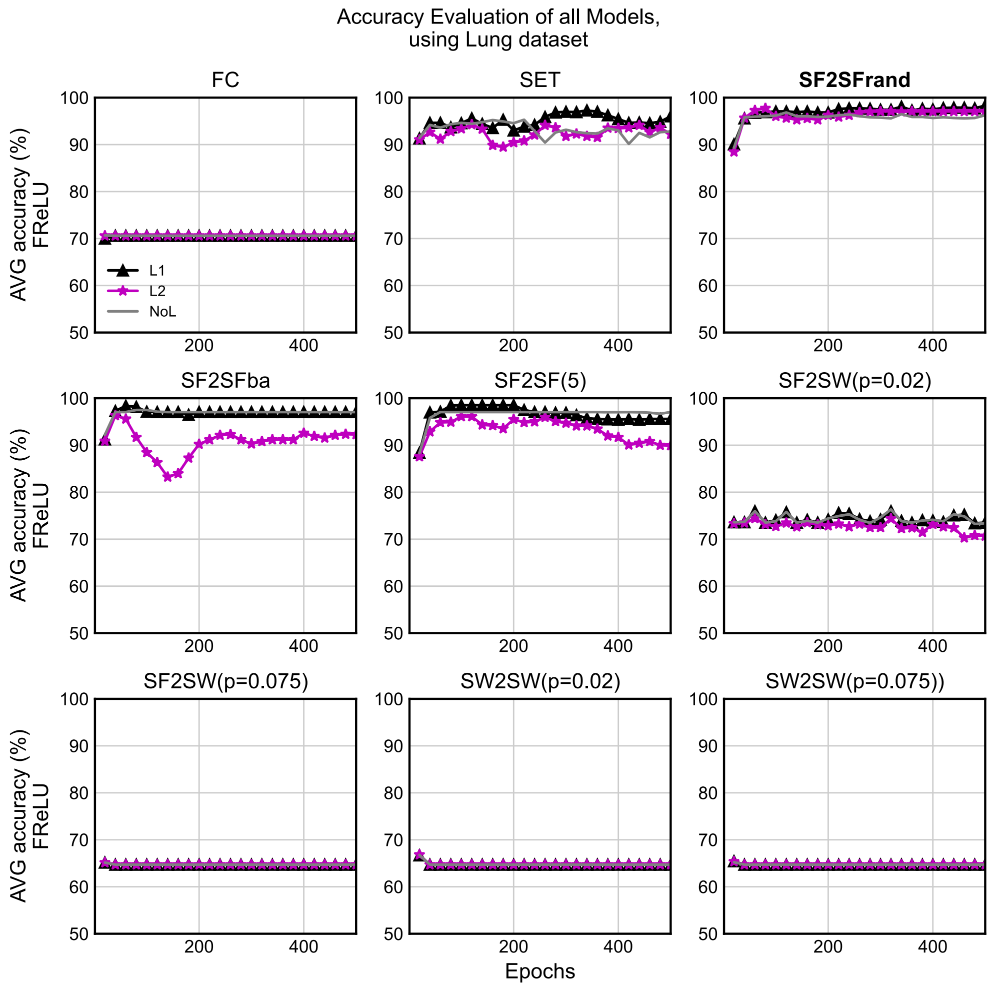


Fig. 13 Results of MLP(FC) and AVG-Weighted algorithms on Lung dataset, using FReLU activation function

We summarize the average accuracy (just before convergence), and the average training times of the algorithms in Figs. 18 and 19. There we can see that SF2SFrand achieves almost equal accuracy to SET, but it does this with almost 30% of its training time.

In Figs. 12 and 13, the accuracy of the baseline method (SET), our best method (SF2SFrand) and the new average-weighted-SF2SFrand(AVG-weighted) algorithm is illustrated, using both ReLU and FReLU activation function. The results (in Tables 11, and 12) shown that there is no crucial difference between these algorithms regarding accuracy but there is so, in the time needed for their training. For example, picking AVG-SF2SFrand algorithm in case of using ReLU activation function and L2 regularization technique, the accuracy is about 0.18% higher than the one achieved by the old implementation of SF2SFrand. However, the time needed for the new implementation of the algorithm to converge is 80.4% larger. Generally, in most of the cases, the accuracy of the AVG-weighted algorithms and the proposed algorithms is approximately the same, while the increase of the training time, needed, when an AVG-weighted algorithm achieves better performance, fluctuates between 18% to 80%. So, taking into consideration that our algorithms must be capable of running on devices with memory constraints, it is imperative for the algorithm to process the training phase as fast as possible. Therefore, our proposed implementations perform better, being applied on resource-constrained devices.

In Figs. 14, and 15 the results regarding the accuracy of SET, SF2SFrand and ζ-parametarized SF2SFrand on Lung and Fashion-Mnist dataset, respectively, are depicted. In this experiment, we initialize ζ parameter to 40% and then, it declines during the training phase, while in the previous algorithms we put ζ value to 30%. We can conclude that ζ-

Table 11 Measures of experiments with AVG weighted algorithms on Lung dataset

Measures - Experiments with AVG weights in Lung.mat dataset

Algorithm-	MLP (FC)	SET	SF2SFrand	Sf2SFba	SF2SF(5)	SF2SW-0.02	SF2SF-0.075	SW2SW-0.02	SW2SW-0.075
Activ.Fun.+Regul.	*Acc/Time*	*Acc/Time*	*Acc/Time*	*Acc/Time*	*Acc/Time*	*Acc/Time*	*Acc/Time*	*Acc/Time*	*Acc/Time*
ReLU + NoL	70.58% /1h 15 min	92.02% /35 min	90% /1h 20 min	90.7% /32 min	89.54% / 29 min	73.79%/4h 16 min	64.72%/6h 49 min	64.78%4h 29 min	64.74%6h 26 min
ReLU + L1	70.58% /2h 22 min	90.53% /47 min	93.7% / 1h 22 min	92.9% /38 min	88.90%/31 min	73.69%/4h 20 min	64.72%6h 54 min	64.78%4h 35 min	64.74%6h 37 min
ReLU + L2	99.52% /1h 23 min	92.71% /38 min	90% /1h 24 min	94.4% /34 min	88.81%/35 min	69.97%/4h 16 min	64.72%6h 47 min	64.79%4h 30 min	64.74%6h 31 min
FReLU + NoL	70.58% /1h 28 min	93.02% /45 min	87% /1h 26 min	96.9% /38 min	96.63%/35 min	74.26%/4h 20 min	64.72%6h 52 min	64.78%4h 31 min	64.74%6h 33 min
FReLU + L1	70.58% /2h	95.04% /53 min	97% /1h 24 min	96.9% /43 min	96.56%/36 min	74.29%4h 26	64.72%6h 54 min	64.78%4h 30 min	64.74%6h 36 min
FReLU + L2	70.58% /1h 26 min	92.38% /49 min	86% /1h 26 min	90.7% /39 min	93.28%/39 min	72.69%/4h 20 min	64.72%6h 54 min	64.79%4h 34 min	64.74%6h 33 min

parametarized SF2SFrand achieved approximately the same or a lower level of accuracy. Also, the last one seems to start stabilizing its accuracy evolution faster than the other two algorithms, when a smaller dataset is used (lung), still, the training time needed for the algorithm to converge is twice as long as the SF2SFrand algorithm (as it is shown in Table 13), since the number of links that the algorithm has to process is much larger. This happens because the number of pruned links is not stable, but it decreases epoch after epoch exponentially. Therefore, the network tends to become less sparse and more similar to the dense network epoch after epoch, since after the 130th epoch, the number of links, being pruned,is very small and this value tends to be stabilized since, ζ fluctuates among 10% and 5%, as it is illustrated in Fig. 16. So, our proposed method - SF2SFrand is also the winner since it achieves comparable accuracy (,even comparing it with the case that uses small values of ζ) regarding the other methods, mentioned. At the same time, it reduces, concurrently, the memory footprint of the network, since it cuts more links than the ζ-parametarized SF2SFrand does.

Furthermore, testing our five proposed algorithms on Fashion-Mnist dataset (which is a large scale dataset) for 100 epochs and ReLU activation function, as it is illustrated in Fig. 17, we can confirm that all our proposed methods retain high accuracy regarding not only the fully-connected model but also the baseline model - SET. The most interesting thing here is that our best method - SF2SFrand, continues to generalize the given data faster than the SET implementation does, while in some cases, achieves about 4.9% higher accuracy than SET. For example, when ReLU activation function and L1 regularization technique are used, the MLP(FC) network needs extremely huge time - about 2 days 16 hours and 51 minutes to be trained, the SET algorithm needs 11 hours and 22 minutes to converge, while our winning method needs 9 hours and 2 minutes to converge, too. We can conclude, that our method achieves approximatelly 20.5% faster convergence than SET does,in most of the cases, as it is shown in Figs. 20 and 21 and in Table 14, in detail. Due to the extended training time needed from the algorithms for their convergence, we test our algorithms with the fastest activation function, having at our disposal, the ReLU (Figs. 18, 19, 20 and 21).

5.3 Summary of experimental results

To sum up, in Table 15, we demonstrate the overall results of our best sparse model training techniques in comparison with both the fully-connected model and the model constructed by SET, which is the only prior work close to ours. We start with a sparse network implemented with a randomly initialized sparse table, and we continue to cut and reconnect links of the network, during the training

Table 12 Comparison between the old and new implementation of all algorithms

Comparison between the old and the new AVG experiment on lung dataset

Algorithms	Old Experiment	New AVG Experiment
Dataset	Activ.Fun.+Regul.	Activ.Fun.+Regul.
Lung.mat	$Acc/Time$	$Acc/Time$
SF2SFrand	FReLU + L2	ReLU+L2
	92.9% /16 min	93.07%/1h 22 min
SF2SFba	FReLU + NoL	FReLU+NoL
	95.6% /32 min	95.5%/38 min
SF2SF(5)	FReLU + NoL	FReLU+NoL
	92.8% /35 min	92.63%/35 min
SF2SW(p=0.02)	ReLU + L1	FReLU+L1
	74.8% /4h 17 min	74.29%/4h 26 min
SF2SW(p=0.075)	ReLU + NoL	ReLU+L2
	64.7% /6h 3 min	64.72%/6h 47 min
SW2SW(p=0.02)	FReLU + NoL	ReLU+L2
	81% /4h 13 min	64.79%/4h 30 min
SW2SW(p=0.075)	FReLU + L1	ReLU+NoL
	65% /5h 50 min	64.74%/6h 26 min

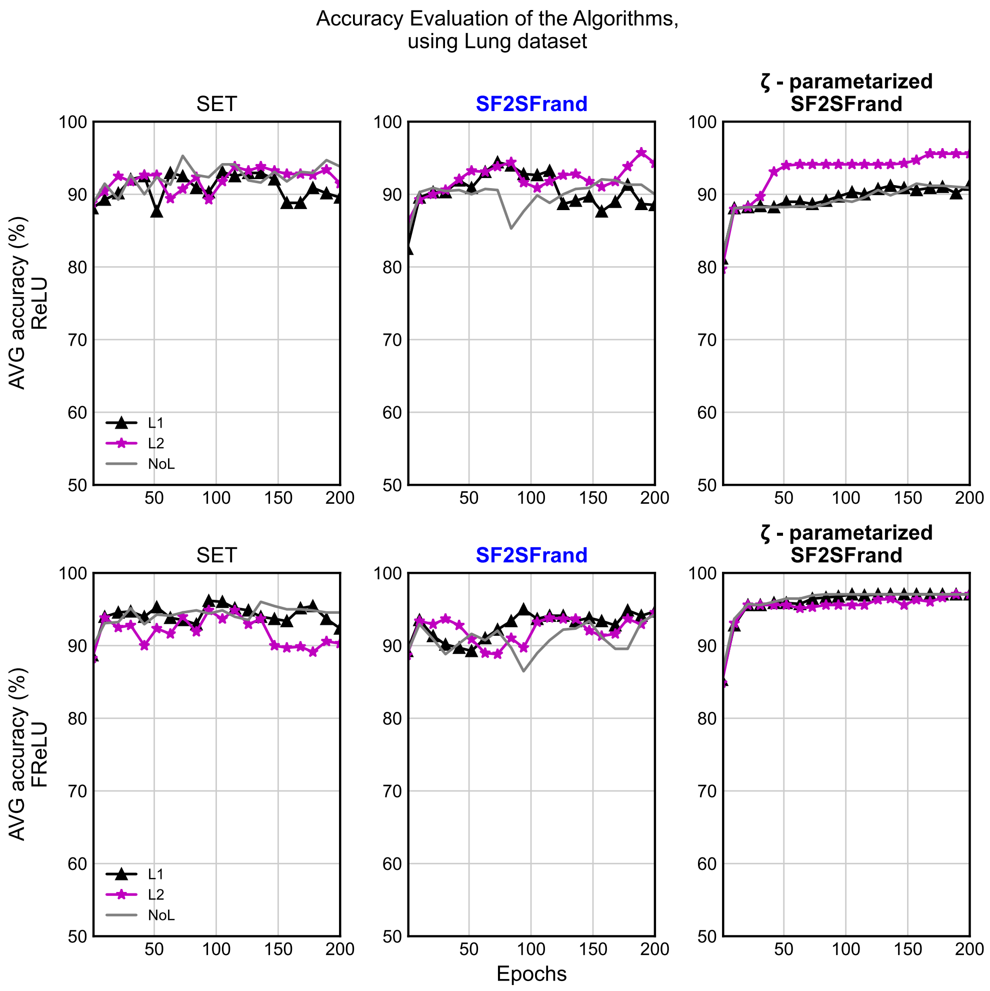


Fig. 14 Results of SET, SF2SFrand ζ-parametarized SF2SFrand, using both ReLU and FRelU activation function and Lung dataset

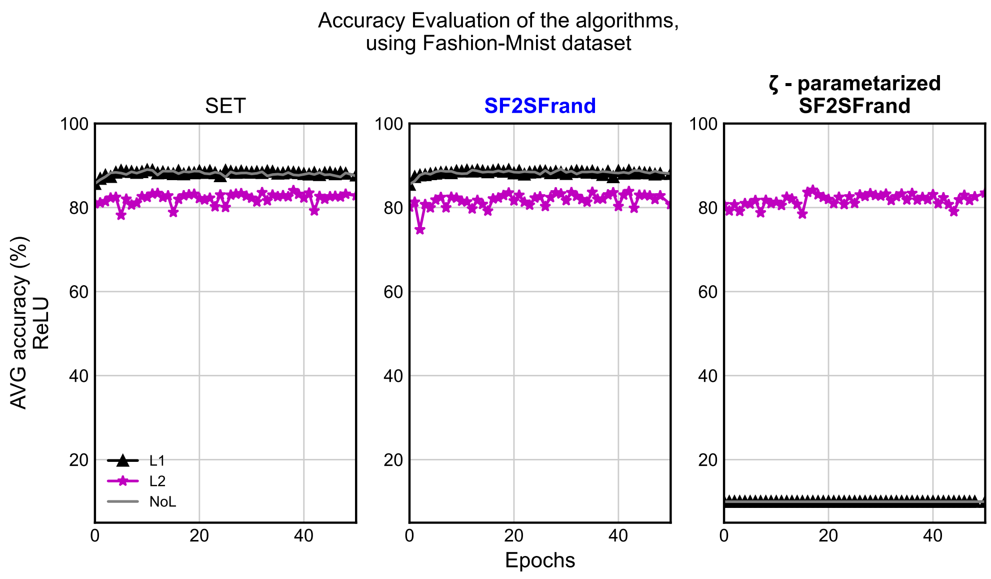


Fig. 15 Results of SET, SF2SFrand ζ-parametarized SF2SFrand, using ReLU activation function and Fashion-Mnist dataset

phase, in order for the model to converge. We see that SF2SFrand is our winning method because it achieves to compress the network capacity by reducing its trainable parameters and consequently, the memory requirements of the network up to 76% in comparison with the memory that the fully connected network needs, and up to 1% regarding the memory that our competitor (SET implementation) requires.

Furthermore, we manage to reduce the training time up to 50% in most of the cases, while we almost preserve the training accuracy of our competitors (for example, SF2SFrand achieves about up to 92.9% training accuracy in 16 minutes, while the dense model achieves about 99.5% accuracy in 1 hour and 23 minutes and the SET implementation achieves 92.71% in 38 minutes when Lung dataset is used), as it is also illustrated in Table 15. It is worth highlighting that our proposed schemes try to sparsify the network by deleting the connections that do not have substantial weights to contribute to the data generalization; concurrently, it reconnects as many connections as the ones being deleted, in a carefully designed way, creating sale free-like topology schemes, so as for dataset information to be distributed effectively and hence, achieve outstanding performance, regarding not only the training time but also the accuracy and the memory footprint.

Table 13 Measures of ζ-parametarized SF2SFrand

Measures - ζ-parametarized SF2SFrand for 40 epochs			
Lung dataset		Fashion-Mnist dataset	
Activ.Fun. +Regul.	$Acc/Time$	Activ.Fun. +Regul.	$Acc/Time$
ReLU+L1	91.17%/25 min	ReLU+L1	88.55%/4h 33 min
ReLU+L2	92.64%26 min	ReLU+L2	83.47%/3h 35 min
ReLU+NoL	95.58%/25 min	ReLU+NoL	88.05% 1d 1h 55 min
FReLU+L1	96.05%/27 min	FReLU+L1	87.28%/9h 7 min
FReLU+L2	96.07%/28 min	FReLU+L2	10%/1d 5h 31 min
FReLU+NoL	96.05%/33 min	FReLU+NoL	10%/3h 33 min

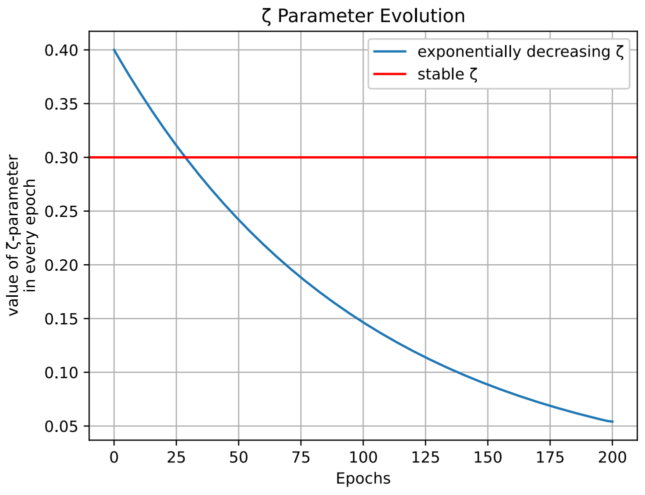


Fig. 16 Evolution of ζ in every epoch

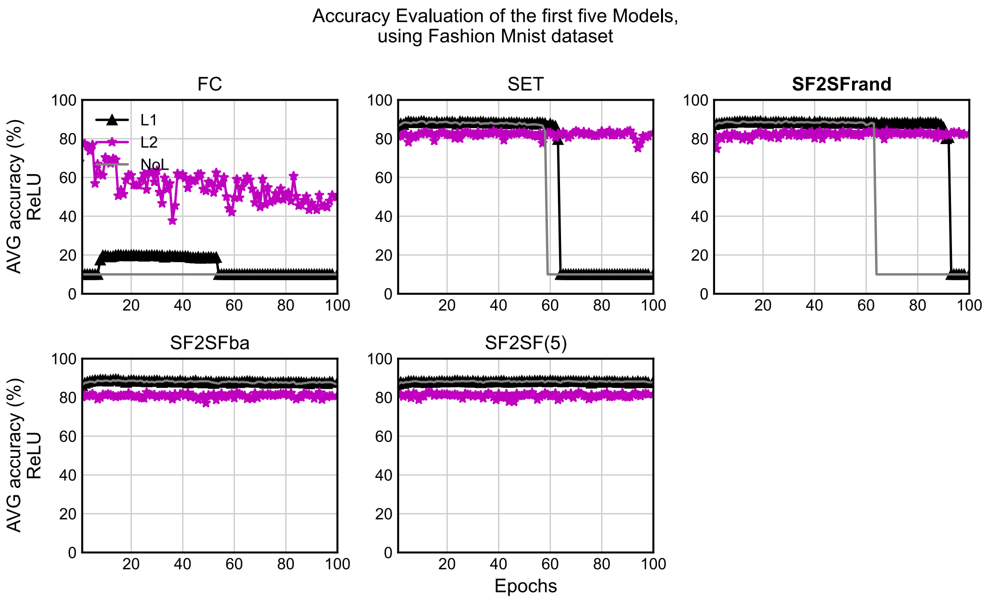


Fig. 17 Overall results on five algorithms, using Fashion-Mnist dataset

Table 14 Measures of MLP(FC), SET, SF2SFrand, SF2SFba, SF2SF(5) algorithms, respectively, in fashion-mnist dataset

Measures - Fashion-Mnist dataset					
Datasets-	MLP (FC)	SET	SF2SFrand	SF2SFba	SF2SF(5)
Activ.Fun.+Regul.	*Acc/Time*	*Acc/Time*	Activ.Fun.+Regul.	*Acc/Time*	*Acc/Time*
ReLU + NoL	10%/5d 1h 14 min	83.74%/11h 22 min	87.86%/ 9h 2 min	86.60%/ 21h 40 min	87.74% /5h 4 min
ReLU + L1	10%/2d 16h 51 min	79.63%/15h 23 min	80.65% /12h 56 min	87.8% / 13h 7 min	88.22%/ 7h 46 min
ReLU + L2	49.94%/2d 13h 5 min	83% /12h 50 min	82.68% 18h 49 min	80.81% /10h 56 min	81.65% /5h 54 min

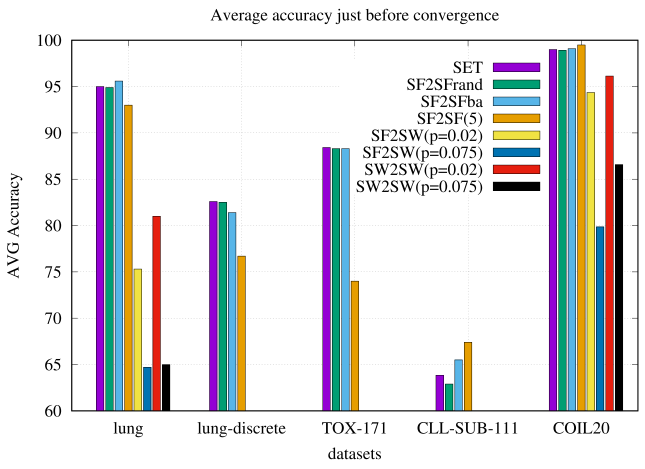


Fig. 18 Accuracy achieved by the competitors for the five datasets

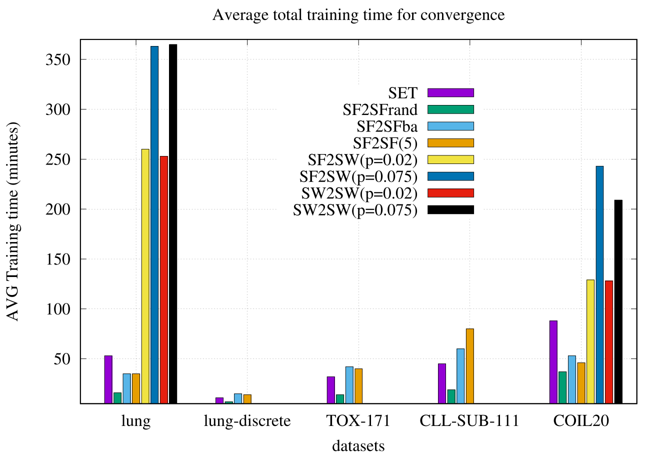


Fig. 19 Execution time of the competitors for the five datasets

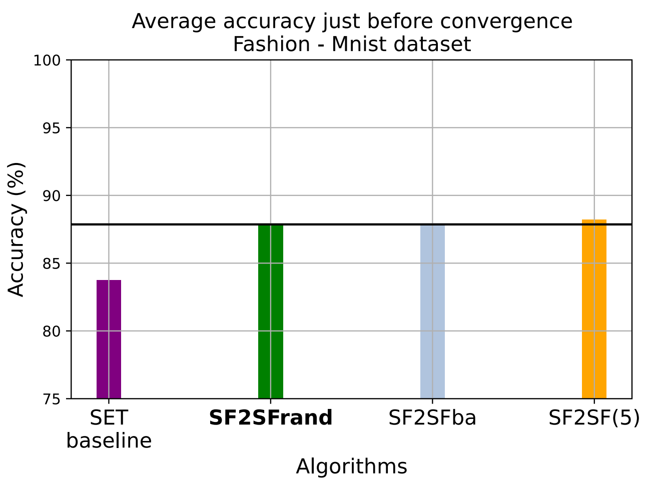


Fig. 20 Accuracy achieved by the comparable algorithms, using Fashion-Mnist dataset

6 Conclusions

The tremendous success of deep learning has brought neural networks to the forefront of machine learning research and development. Due to the large size of a neural network - in the number of neurons and the number of hidden layers - training the network in a relative short time is a challenge. Various methods have been developed for accelerating neural training over the past thirty years. We focus here on the family of methods based on linkage sparsification, i.e., instead of fully connected bipartite neural topologies, we reduce the number of connections in an algorithmic (or random) way. In particular, we employ concepts developed in the realm of network science, in order to sparsify the neural network with the aim of keeping the most significant linkages, which are responsible for better information distribution in the network. Thus, we reduce

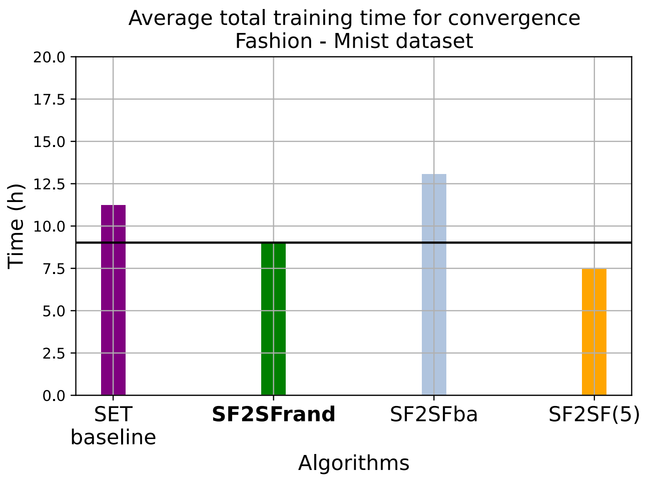


Fig. 21 Execution time of the comparable algorithms, using Fashion-Mnist dataset

Table 15 Summary of experiments

Summary of experiments								
Algorithm	Memory Footprint	Percentage Reduction in Memory	Lung Acc /Time	Lung Discrete Acc/Time	TOX-171 Acc/Time	CLL-SUB-111 Acc/Time	COIL20 Acc/Time	Fashion-Mnist Acc/Time
MLP (FC)	77.64 MB	—	99.52% /1h 23min	80% /5 min	24.56% /75 min	19.07% /1h 5 min	62.5% /6h 3 min	49.94%/2d 13h 5 min
SET	19.18 MB	75.2%	92.71% /38 min	82.59% /11 min	88.43% /32 min	65.84% /45 min	99.37% /1h 28 min	83.74%/11h 22 min
SF2SFrand	18.39 MB	76.31%	92.9% /16 min	82.5% /7 min	83.3% /14 min	62.4% /19 min	99.16% /1h 42 min	87.86%/9h 2 min
SF2SFba	20.68 MB	73.36%	95.6% /32 min	82.4% /15 min	95% /42 min	65.5% /1h	99.16% /53 min	87.8%/13h 7 min
SF2SF(5)	20.18 MB	74.00 %	92.8% /35 min	90.6% /14 min	81% /40 min	70.25% /1h 13 min	99.79% /1h 13 min	88.22%7h 46 min

drastically both the number of trainable variables and the size of the model, which leads to 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-sized neural network in a publicly available dataset and examined their classification accuracy and training time. We concluded that the proposed techniques can reap performance gains, achieving high accuracy with a short training time. The 'champion' algorithm was the one that produced scale-free-like topologies starting from scale-free topologies. Specifically, the SF2SFrand algorithm outperforms every other method in most of the cases mentioned, achieving high percentages of both memory (approximately 76.31% less than FC-MLP dense model and 3% less than the baseline method - SET) and training time (about 50% less in most of the cases, compared to baselines(FC-MLP, SET) and 20.5% less in the experiments with large scale datasets, like Fashion-Mnist) reduction and, while retaining classification accuracy of baseline methods. 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 approaches [10, 34] to the problem of neural training acceleration.

Acknowledgements Part of this work was done in the context of the BSc thesis/dissertation (2019) of the first two authors in the University of Thessaly, entitled "Neural Network Training Techniques Based on Topology Sparsification" and "Speeding up Neural Network Training via Topology Sparsification".

Declarations

The authors declare that they have no known conflicting/competing financial or non-financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Barabasi A-L (2016) Network Science Cambridge University Press
2. Barabasi A-L, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512
3. Basaras P, Katsaros D, Tassiulas L (2013) Detecting influential spreaders in complex, dynamic networks. IEEE Comp Magazine 46(4):26–31
4. Bullmore E, Sporns O (2009) Complex brain networks: graph theoretical analysis of structural and functional systems. Nature Rev Neuroscience 10:186–198
5. Cai H, Gan C, Zhu L, Han S (2020) TinyTL: reduce memory, not parameters for efficient on-device learning. In: Proceedings of the conference on neural information processing systems (NeurIPS
6. Cavallaro L, Bagdasar O, Meo PD, Fiumara G, Liotta A (2020) Artificial neural networks training acceleration through network science strategies. Soft Comput 24:17787–17795
7. Chouliaras A, Fragkou E, Katsaros D (2021) Feed forward neural network sparsification with dynamic pruning. In: Proceedings of the panhellenic conference on informatics (PCI)
8. Diao H, Li G, Hao Y (2022) PA-NAS: partial operation activation for memory-efficient architecture search. Appl Intell. To appear
9. Erkaymaz O (2020) Resilient back-propagation approach in small-world feed-forward neural network topology based on newman-watts algorithm. Neural Comput Applic 32:16279–16289
10. Frankle J, Carbin M (2019) The lottery ticket hypothesis: finding sparse, trainable neural networks. In: Proceedings of the international conference on learning representations (ICLR)
11. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. The MIT Press
12. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Proceedings of advances in neural information processing systems, pp 1135–1143
13. Han S, Mao H, Dally WJ (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Proceedings of the international conference on learning representations (ICLR
14. Hao J, Cai Z, Li R, Zhu WW (2021) Saliency: a new selection criterion of important architectures in neural architecture search. Neural Comput Appl. To appear
15. Hoefler T, Alistarh D, Ben-Nun T, Dryden N, Peste A (2021) Sparsity in deep learning pruning and growth for efficient inference and training in neural networks. J Mach Learn Res 23:1–124
16. Hong Z-Q, Yang J-Y (1991) Optimal discriminant plane for a small number of samples and design method of classifier on the plane. Pattern Recogn 24:317–324
17. Iiduka H (2022) Appropriate learning rates of adaptive learning rate optimization algorithms for training deep neural networks. IEEE Trans Cybern. To appear
18. James AP, Dimitrijev S (2012) Feature selection using nearest attributes. Available at: arXiv:1201.5946
19. Jouppi NP, Young C, Patil N, Patterson D (2018) Domain-specific architecture for deep neural networks. Commun ACM 61(9):50–59
20. Liebenwein L, Baykal C, Carter B, Gifford D, Rus D (2021) Lost in pruning: the effects of pruning neural networks beyond test accuracy. In: Proceedings of the machine learning systems conference (MLSys
21. Liu S, Mocanu DC, Matavalam ARR, Pei Y, Pechenizkiy M (2020) Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. Neural Comput Applic 33:2589–2604
22. Mocanu DC, Mocanu E, Stone P, Nguyen PH, Gibesce M, Liotta A (2018) Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat Commun, pp 9
23. Mokhtari A, Ribeiro A (2015) Global convergence of online limited memory BFGS. J Mach Learn Res 16:3151–3181
24. Narang S, Diamos G, Sengupta S, Elsen E (2017) Exploring sparsity in recurrent neural networks. In: Proceedings of the international conference on learning representations (ICLR)
25. Nene SA, Nayar SK, Murase H (1996) Columbia object image library (COIL-20). Technical report CUCS-006-96 Columbia University

26. Papakostas D, Kasidakis T, Fragkou E, Katsaros D (2021) Backbones for internet of battlefield things. In: Proceedings of the IEEE/IFIP annual conference on wireless on-demand network systems and services (WONS)
27. Qiu S, Xu X, Cai B (2019) FReLU: flexible rectified linear units for improving convolutional neural networks. Available at arXiv:1706.08098
28. Ray PP (2022) A review on tinyML: state-of-the-art and prospects. J King Saud University– Comput Inf Sci, To appear
29. Reddi SJ, Kale S, Kumar S (2018) On the convergence of Adam and beyond. In: Proceedings of the international conference on learning representations (ICLR)
30. Ren P, Xiao Y, Chang X, Huang P-Y, Li Z, Chen X, Wang W (2021) A comprehensive survey of neural architecture search challenges and solutions. ACM Comput Surv 54(76):1–34
31. Renda A, Frankle J, Carbin M (2020) Comparing rewinding and fine-tuning in neural network pruning. In: Proceedings of the international conference on learning representations (ICLR)
32. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
33. Sun X, Ren X, Ma S, Wang H (2017) meProp: sparsified back propagation for accelerted deep learning with reduced overfitting. Proc Mach Learn Res 70:3299–3308
34. Sun X, Ren X, Ma S, Wei B, Li W, Xu J, Wang H, Zhang Y (2019) Training simplification and model simplification for deep learning: a minimal effort back propagation method. IEEE Trans Kowl Data Eng, A minimal effort back propagation method. IEEE Transactions on Kowledge and Data Engineering, Training simplification and model simplification for deep learning. To appear
35. Wang X, Zheng Z, He Y, Yan F, qiang Zeng Z, Yang Y (2021) Soft person reidentification network pruning via blockwise adjacent filter decaying. IEEE Trans Cybern
36. Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747
37. Xu S, Chen H, Gong X, Liu K, Lu J, Zhang B (2021) Efficient structured pruning based on deep feature stabilization. Neural Comput Applic 33:7409–7420
38. Zlateski A, Lee K, Seung HS (2017) Scalable training of 3d convolutional networks on multi- and many-cores. J Parallel Distrib Comput 106:195–204

Evangelia Fragkou received her diploma (Integrated Master's Degree + BEng+MEng) from the Department of Electrical and Computer Engineering, at Volos, in 2019. She is now a PhD candidate at the same Department. She has earned an ELIDEK grant to carry our her PhD studies. Her main research interests lie in the area of machine/deep learning.

Marianna Koultouki received her diploma (Integrated Master's Degree + BEng+MEng) from the Department of Electrical and Computer Engineering, at Volos, in 2019. She now works as a software engineer on projects related to object-oriented programming and machine learning tasks.

Dimitrios Katsaros is an associate professor with the Department of Electrical and Computer Engineering at the University of Thessaly. During the fall semester of 2022 he is on sabbatical leave of absence cooperating with the Carnegie Mellon University at Qatar. He has been a visiting fellow (2015) and a visiting assistant professor (2017) in the Department of Electrical Engineering at Yale University and also with the Yale Institute for Network Science; he has been a visiting professor (2019) in KIOS Research and Innovation Centre of Excellence at the University of Cyprus His research interests include distributed systems and algorithms. Katsaros received a PhD in informatics from the Aristotle University of Thessaloniki, Greece in 2004.