Noname manuscript No.
(will be inserted by the editor)

A Brain-inspired Algorithm for Training Highly Sparse Neural Networks

Zahra Atashgahi · Joost Pieterse · Shiwei Liu · Decebal Constantin Mocanu · Raymond Veldhuis · Mykola Pechenizkiy

Received: date / Accepted: date

Abstract Sparse neural networks attract increasing interest as they exhibit comparable performance to their dense counterparts while being computationally efficient. Pruning the dense neural networks is among the most widely used methods to obtain a sparse neural network. Driven by the high training cost of such methods that can be unaffordable for a low-resource device, training sparse neural networks sparsely from scratch has recently gained attention. However, existing sparse training algorithms suffer from various issues, including poor performance in high sparsity scenarios, computing dense gradient information during training, or pure random topology search. In this paper, inspired by the evolution of the biological brain and the Hebbian learning theory, we present a new sparse training approach that evolves sparse neural networks according to the behavior of neurons in the network. Concretely, by exploiting the cosine similarity metric to measure the importance of the connections, our proposed method, "Cosine similarity-based and Random Topology Exploration (CTRE)", evolves the topology of sparse neural networks by adding the most important connections to the network without calculating dense gradient in the backward. We carried out different experiments on eight datasets, including tabular, image, and text datasets, and demonstrate that our proposed method outperforms several state-of-the-art sparse training algorithms in extremely sparse neural networks by a large gap. The implementation code is available on Github[1].

Z. Atashgahi · D.C. Mocanu · R.N.J. Veldhuis
Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500AE Enschede, the Netherlands
Tel.: +31534896141
E-mail: z.atashgahi@utwente.nl

J. Pieterse · S. Liu · D.C. Mocanu · M. Pechenizkiy
Faculty of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, the Netherlands

M. Pechenizkiy
Faculty of Information Technology, University of Jyväskylä, 40014 Jyväskylä, Finland

[1] `https://github.com/zahraatashgahi/CTRE`

1 Introduction

Dense artificial neural networks are a commonly used machine-learning technique that has a wide range of application domains, such as speech recognition [18], image processing [41, 50], and natural language processing (NLP) [6]. It has been shown in [26] that the performance of deep neural networks scales with model size and dataset size, and generalization benefits from over-parameterization [58]. However, the ever-increasing size of deep neural networks has given rise to major challenges, including high computational cost both during training and inference and high memory requirement [76]. Such an increase in the number of computations can lead to a critical rise in the energy consumption in data centers and, consequently, a deteriorative effect on the environment [75]. However, a trustworthy AI system should function in the most environmentally friendly way possible during development, and deployment [19]. In addition, such gigantic computational costs will lead to a situation where on-device training and inference of neural network models on low-resource devices, e.g., an edge device with limited computational resources and battery life, might not be economically viable [76].

Sparse neural networks have been considered as an effective solution to address these challenges [27, 53]. By using sparsely connected layers instead of fully-connected ones, sparse neural networks have reached a competitive performance to their dense equivalent networks in various applications [12, 3], while having much fewer parameters. It has been shown that biological brains, especially the human brain, enjoy sparse connections among neurons [13]. Most existing solutions to obtain sparse neural networks focus on inference efficiency in order to reduce the storage requirement of deploying the network and prediction time of test instances. This class of methods, named *dense-to-sparse* training, starts by training a dense neural network followed by a pruning phase that aims to remove unimportant weight from the network. As categorized in [53], in dense-to-sparse training, the pruning phase can be done after training [37, 12, 23], simultaneous to training [48], or one-shot prior to training [38]. However, starting from a dense network leads to a memory requirement of fitting a dense network on the device and the computational resources for at least a few iterations of training the dense model. Therefore, training sparse neural networks using dense-to-sparse methods might be infeasible on low-resource devices due to the energy and computational resource constraints.

With the emergence of the *sparse training* concept in [51], there has been a growing interest in training sparse neural networks which are sparse from scratch. This sparse connectivity might be fixed during training (known as static sparse connectivity [51, 53, 31]), or might dynamically change, by removing and re-adding weights (known as dynamic sparse connectivity [52, 5]). By optimizing the topology along with the weights during the training, dynamic sparse training algorithms outperform the static ones [52]. As discussed in [52], the weight removal in dynamic sparse training algorithms is similar to the synapses shrinkage in the human brain during sleep, where the weak synapses shrink and the strong ones remain unchanged. While most dynamic sparse training methods use magnitude as a pruning criterion, weight regrowing approaches are of different types, including random [52, 57] and gradient-based regrowth [10, 28]. As shown in [47], random addition of weights might lead to a low training speed, and the performance of sparse training is highly correlated with the total number of parameters explored during training. To speed up the convergence, gradient information of non-existing connections can be used to add the most important connections to the network [9]. However, computing the gradient of all non-existing connections in a sparse neural network can be computationally demanding. Furthermore, increasing the network size might escalate the high computational cost into a bottleneck in the sparse training of networks on low-resource devices. Besides, in Section 4.2, we demonstrate that some gradient-based sparse training algorithms might fail in a highly sparse neural network.

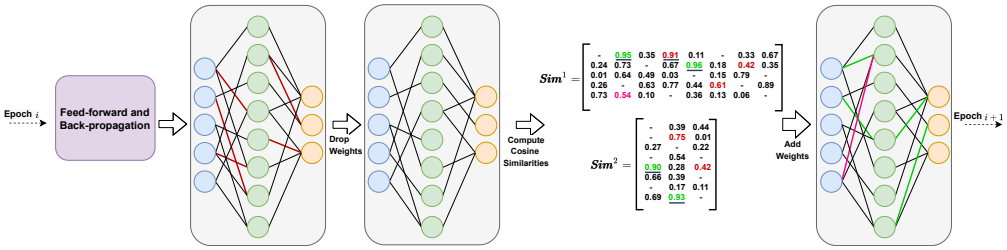

Fig. 1: Schematic of the proposed approach (CTRE_{sim}). At each epoch, after feed-forward and back-propagation, a fraction ζ of the weights with the smallest magnitude is dropped (red connections). Then, similarity matrices $\boldsymbol{Sim}^1$ and $\boldsymbol{Sim}^2$ are computed using Equation 2 to find the most important connections to add to the network; however, we do not consider the similarity of the existing connections (empty entries). Finally, the weights corresponding to the highest similarity values in the similarity matrices (underlined values) that have not been dropped in the weight removal step are added to the network (underlined green values), the same amount as removed previously. If a connection with high similarity has been dropped in the weight removal step (underlined red value), a random connection will be inserted instead (pink connection).

In this paper, to address some of these challenges, we introduce a more biologically plausible algorithm for obtaining a sparse neural network. By taking inspiration from the Hebbian learning theory, which states "neurons that fire together, wire together" [25], we introduce a new weight addition policy in the context of sparse training algorithms. Our proposed method, "Cosine similarity-based and Random Topology Exploration (CTRE)", exploits both the similarity of neurons as an importance measure of the connections and random search simultaneously (CTRE_{sim}, Figure 1) or sequentially (CTRE_{seq}) to find a performant sub-network. In short, our contributions are as follows:

- We propose a novel and biologically plausible algorithm for training sparse neural networks, which has a limited number of parameters during training. Our proposed algorithm, CTRE, exploits both similarity of neurons and random search to find a performant sparse topology.
- We introduce the Hebbian learning theory in the training of the sparse neural networks. Using the cosine similarity of each pair of neurons in two consecutive layers, we determine the most important connections at each epoch during sparse training of the network; we discuss in detail why this approach is an extension to the Hebbian learning theory in Section 3.2.
- Our proposed algorithms outperform state-of-the-art sparse training algorithms in highly sparse neural networks.

While deep learning models have shown great success in vision and NLP tasks, these models have not been fully explored in the domain of tabular data [61]. However, designing deep models that are capable of processing tabular data is of great interest for researchers as it paves the way to building multi-modal pipelines for problems [17]. This paper mainly focuses on Multi-Layer Perceptrons (MLPs), which are commonly used for tabular and biological data. Despite the simple structure of MLPs and having only a few hyperparameters to tune, they have shown good performance in classification tasks [15, 69]. In addition, in [29],

authors investigated that despite the massive attention on CNN architectures, they utilize only 5% of the neural network workload of TPUs in Google data centers, while MLPs constitutes 61% of the total workload. Therefore, it is crucial to develop an efficient algorithm that can accelerate MLPs and are resource-efficient during training and inference. To pursue this goal, in this research, we aim to design sparse MLPs with a limited number of parameters during training and inference. To demonstrate the validity of our proposed algorithm, in addition to evaluating the methods on tabular and text datasets, we compare the methods also on the image datasets such as MNIST, Fashion-MNIST, and CIFAR10/100 datasets which are commonly used as benchmarks in previous studies.

2 Background

2.1 Sparse Neural Networks

Methods to obtain and train sparse neural networks can be stratified into two major categories: dense-to-sparse and sparse-to-sparse. In the following, we shed light on each of these two approaches.

Dense-to-sparse. Dense-to-sparse methods to obtain sparse neural networks start training from a dense model and then prune the unimportant connections. They can be divided into three major subcategories: (1) *Pruning after training*: Most existing dense-to-sparse methods start with a trained dense network and iteratively (one or several iterations) prune and retrain the network to reach desired sparsity level. Seminal works were performed in the 1990s in [37, 24], where authors use hessian matrix information to prune a trained dense network. More recently, in [23, 12], authors use magnitude to remove unimportant connections. Other metrics, such as gradient [42], Taylor expansion [55, 56], and low-rank decomposition [70, 40], have been also employed to prune the network. While being effective techniques in terms of the performance of the obtained sparse network, these methods suffer from high computational costs during training. (2) *Pruning during training*: To decrease the computational cost, this group of methods perform pruning during training [14, 30, 34]. Various criteria can be used for pruning, such as magnitude [20, 77], L_0 regularization [48, 63], group Lasso regularization [72], and variational dropout [54]. (3) *Pruning before training*: The first study to apply pruning prior to training was done by Lee et al. in [38], that used connection sensitivity to remove weights. Later works have followed the same approach by pruning the network before training using different approaches, such as gradient norm after pruning [71], connection sensitivity after pruning [8], and Synaptic Flow [68].

Sparse-to-sparse. To lower the computational cost of dense-to-sparse methods, sparse-to-sparse training algorithms (also known as sparse training) use a sparse network from scratch with a sparse connectivity, which might be static (static sparse training [51, 31]) or dynamic (dynamic sparse training (DST) [52, 5]). By allowing the topology to be optimized along with the weights, sparse neural networks trained with DST have reached a comparable performance to the equivalent dense networks or even outperform them. DST methods can be divided into two main categories based on the weight addition policy: (1) *Random regrowth*: Sparse Evolutionary Training (SET) [52] is one of the earliest works that starts with a sparse neural network and perform magnitude pruning and random weight regrowing at each epoch to update the topology. In [57], the authors proposed the idea of parameter reallocation automatically across layers during sparse training in CNNS. Many works have further studied sparse training concept recently [47, 16, 45, 3, 46, 44, 47]. (2) *Gradient information*: A group of works have tried to exploit gradient information to speed up the training process in DST

[62]. Dettmers and Zettlemoyer [9] used the momentum of the non-existing connections as a criterion to grow weights instead of random addition in the SET algorithm; While being effective in terms of the accuracy, this method requires computing gradients and updating the momentum for all non-existing parameters. The Rigged Lottery (RigL) [10] addressed the high computational cost by using infrequent gradient information. However, it still requires the computational cost for computing the periodic dense gradients. [28] tried to further improve RigL by using the gradient for only a subset of non-existing weights. In [7], authors exploit gradient information in the search for a performant sub-network and discuss that gradient-based weight addition is biologically plausible.

2.2 Hebbian Learning Theory

The Hebbian learning rule was proposed in 1949 by Hebb as the learning rule for neurons [25] inspired by biological systems. It describes how the neurons' activations influence the connections among them. The classical Hebb's rule indicates "neurons that fire together, wire together". This can be formulated as $\Delta w_{ij} = \eta p_i q_j$, where Δw_{ij} is the change in synaptic weight w_{ij} between two neurons p_i (presynaptic) and q_j (postsynaptic) in two consecutive layers, and η is the learning rate. While some previous works have adapted Hebb's rule to some machine learning tasks, [64, 43], it has not been vastly investigated in many others, particularly in the sparse neural networks. By adapting Hebb's rule to artificial neural networks, we can obtain powerful models that might be close to the function of structures found in neural systems of various species [33]. In [2], authors have incorporated the Hebbian learning theory to train a newly introduced neural network. In [67], the Hebbian learning concept has been used to sparsify the neural networks for face recognition; they drop the connections between the weakly correlated neurons. In [7], authors proposed a gradient-based algorithm for obtaining a sparse neural network; they discuss the gradient-based connection growth policy is mathematically close to the Hebbian learning theory. In this work, by taking inspiration from the Hebbian learning theory, we aim to introduce a new sparse training algorithm for obtaining sparse neural networks.

2.3 Cosine Similarity

In most machine learning problems, the Euclidean distance is a common tool to measure the distance due to its simplicity. However, the Euclidean distance is highly sensitive to the vectors' magnitude [73]. Cosine similarity is another metric that addresses this issue; it measures the similarity of the shapes of two vectors as the cosine of the angle between them. In other words, it determines whether the two vectors are pointing in the same direction or not [22]. Due to its simplicity and efficiency, the cosine similarity is a widely used metric in machine learning and pattern recognition field [73]. It often measures the document similarity in natural language processing tasks [66, 39]. Cosine Similarity has proven to be an effective tool also in neural networks. In [49], to bound the pre-activations in a multi-layer neural network that might disturb the generalization, authors have proposed to use cosine similarity instead of the dot product and showed that it reaches a better performance than the simple dot product. In [59], authors have used this metric to improve face verification using deep learning.

3 Proposed Method

In this section, we first formulate the problem. Secondly, we demonstrate the cosine similarity as a tool for determining the importance of weights in neural networks and how it relates to the Hebbian learning theory. Finally, we present two new sparse training algorithms using cosine similarity-based connection importance.

3.1 Problem Definition

Given a set of training samples $\mathbb{X}$ and target output $\boldsymbol{y}$, a dense neural network is trained to minimize $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})$, where m is the number of training samples, L is the loss function, f is a neural network parametrized by $\boldsymbol{\theta}$, $f(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$ is the predicted output for input $\boldsymbol{x}^{(i)}$, and $\boldsymbol{y}^{(i)}$ is the true label. $\boldsymbol{\theta} \in \mathbb{R}^N$ is consisted of parameters of each layer $l \in \{1,2,...,H\}$ of the network as $\boldsymbol{\theta}^l \in \mathbb{R}^{N^l}$, where $N^l = n^{l-1} \times n^l$ is the number of parameters of layer l, n^l is number of neurons at layer l, and the total number of parameters of the dense network is N. A sparse neural network, however, uses only a subset of $\boldsymbol{\theta}^l$, and discards s^l fraction of parameters of each layer $\boldsymbol{\theta}^l$ (their weight values are equal to zero); s^l is referred to as the *sparsity* of layer l. The overall sparsity of the network is $S = 1 - D$, where $D = \frac{\sum_{l=1}^{H}(1-s^l)N^l}{N}$ is the overall density of the network. We aim to obtain a sparse neural network with sparsity level of S and parameters $\boldsymbol{\theta}$. We aim to train this network to minimize the loss on the training set as follows:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\mathbb{R}^N,\ \|\boldsymbol{\theta}\|_0 = D\times N}{\arg\min} \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)}), \tag{1}$$

where $\|\boldsymbol{\theta}\|_0$ is the total number of non-zero connections of the network which is determined by the density level.

Network Structure. The architecture we consider is a Multi-layer Perceptron (MLP) with H layers. Initially, sparse connections between two consecutive layers are initialized with an Erdős–Rényi random graph; each connection in this graph exists with a probability of $P(\theta_i^l) = \frac{\varepsilon(n^{l-1}+n^l)}{n^{l-1}n^l}$, $i \in \{1,2,...,N^l\}$, where ε denotes the hyperparameter that controls the sparsity level. Each existing connection is initialized with a small value from a normal distribution.

3.2 Cosine Similarity to Determine Connections Importance

In this paper, we use the cosine similarity as a metric to derive the importance of non-existing connections and evolve the topology of a sparse neural network. We first demonstrate how do we measure cosine similarity of two neurons. Then, we argue why this choice has been made and how it relates to the Hebbian Learning theory. We measure the similarity of two neurons p and q as:

$$Sim^l_{p,q} = \left| \frac{\boldsymbol{A}^{l-1}_{:,p} \cdot \boldsymbol{A}^{l}_{:,q}}{\left\|\boldsymbol{A}^{l-1}_{:,p}\right\| \left\|\boldsymbol{A}^{l}_{:,q}\right\|} \right|, \tag{2}$$

where $\boldsymbol{Sim}^l$ is the similarity matrix between neurons in two successive layers $l-1$ and l. $\boldsymbol{A}^{l-1}_{:,p}$ and $\boldsymbol{A}^{l}_{:,q} \in \mathbb{R}^m$ are the activation vectors corresponding to neurons p and q in layers $l-1$ and l, respectively. If $Sim^l_{p,q}$ is high for two unconnected neurons (close to 1), it means that they have a high similarity among their activations; therefore, we prefer to add a connection

between them as it suggests that this path contains important information about data. However, if $Sim^{l}_{p,q}$ is low for two neurons (close to 0), it means that the activations of neurons p and q are not similar, and the connection among them might not be beneficial for the network.

We now argue why cosine similarity can be used to measure the importance of a non-existing connection in sparse neural networks and how it connects to the Hebbian learning theory. Basically, by taking inspiration from the Hebbian learning theory, we aim to rewire the neurons that fire together in the context of sparse training algorithms, instead of only strengthening the existing connections among neurons that fire together [65]. It has been discussed in [65] that connecting a pair of neurons with strong coincident activations can be viewed as a natural extension of the Hebbian learning; it is necessary to wire the neurons that usually fire together in order to understand better the relationship among the higher-order representation of those neurons. If a causal connection between their higher-order representation does exist, growing a connection among them will enable an effective inference about the relationship between them. Therefore, we need to discover which pairs of neurons usually fire together and then rewire them.

We employ cosine similarity to measure the relation between the activation values of two neurons. Such as the Hebb's rule (Section 2.2), the importance of a connection in our method is also determined by multiplying the activations of its corresponding neurons, albeit normalized; in Equation 2, $\boldsymbol{A}^{l-1}_{:,p}$ is the presynaptic activation and $\boldsymbol{A}^{l}_{:,q}$ is the postsynaptic activation. If the activations of two connected neurons agree, by computing the dot product of activations, both Equation 2 and Hebb's rule assign higher importance to the corresponding connection. This would result in increased weight and a better chance of adding this connection. Thus, both methods reward connections between neurons that exhibit similar behavior. As mentioned earlier, the main difference between the Hebb's rule and Equation 2 is normalization. We will discuss in Section 5.3 why the normalization step is necessary for evolving the topology of a sparse neural network.

In summary, if the cosine similarity of the activation vector of two neurons is high, it indicates the necessity of the connection between them in the network's performance. Therefore, we use the cosine similarity information to find out if the link between a pair of neurons should be rewired or not. Based on this knowledge, we propose two new algorithms to evolve the sparse neural network in the following sections.

3.3 Sequential Cosine Similarity-based and Random Topology Exploration (CTRE_{seq})

Our first proposed algorithm, Sequential Cosine Similarity-based and Random Topology Exploration (CTRE_{seq}) evolves the network topology using both cosine similarity between neurons of each pair of consecutive layers in the network and random search. Overall, in the beginning, at each training epoch, it removes unimportant connections based on their magnitude and adds new connections to the network based on their cosine similarity. When the network performance stops improving, the algorithm switches to random topology search. In the following, we will explain the algorithm in more detail.

After initializing the sparse network with sparsity level determined by ε, the training begins. The training procedure consists of two consecutive phases: *1. Cosine Similarity-based Exploration*: The training starts with this phase in which each epoch includes three steps: (a) Firstly, a standard feed-forward and back-propagation are performed. (b) Then, a proportion ζ of connections with the lowest magnitude in each layer is removed. In Section 5.2, we further discuss why this choice has been made. (c) Subsequently, we add new connections to the network based on the neurons' similarity. Taking advantage of the cosine similarity metric,

Algorithm 1 CTRE$_{\text{seq}}$

Input: Dataset $\mathbb{X}$, sparsity hyperparameter ε, drop fraction ζ, early stop epoch $e_{early\ stop}$
Initialize the network with sparsity determined by ε, $flag_{random\ search} = False$
for $i \in \{1, \ldots, \#epochs\}$ **do**
  perform standard feed-forward and back-propagation
  for $l \in \{1, \ldots, H\}$ **do**
    Remove ζN^l of the weights with smallest magnitude.
    if $flag_{random\ search}$ **then**
      Add ζN^l connections randomly
    else
      Compute Similarity matrix $\boldsymbol{Sim}^l$ according to Equation 2
      Add ζN^l connections with the highest similarity value in $\boldsymbol{Sim}^l$
  if Accuracy on validation set does not improve in $e_{early\ stop}$ **then**
    Set $flag_{random\ search} = True$

Algorithm 2 CTRE$_{\text{sim}}$

Input: Dataset $\mathbb{X}$, sparsity hyperparameter ε, drop fraction ζ
Initialize the network with sparsity determined by ε
for $i \in \{1, \ldots, \#epochs\}$ **do**
  perform standard feed-forward and back-propagation
  for $l \in \{1, \ldots, H\}$ **do**
    Remove ζN^l of the weights with smallest magnitude.
    Compute Similarity matrix $\boldsymbol{Sim}^l$ according to Equation 2
    C_{sim} = Set of ζN^l connections with the highest similarity value in $\boldsymbol{Sim}^l$
    for each $c \in C_{sim}$ **do**
      if c was removed in the last weight removal step **then**
        Add a random connection to the network
      else
        Add connection c to the network

we measure the similarity of two neurons as formulated in Equation 2. In each layer, we add connections (the same amount as removed previously) with the highest similarity between the corresponding neurons; the new connections are initialized with a small value from a uniform distribution. *2. Random Exploration*: The second phase begins when the performance of the network on a validation set does not improve in $e_{early\ stop}$ epochs ($e_{early\ stop}$ is a hyperparameter of CTRE$_{\text{seq}}$). This is due to the fact that the activation values might not change significantly after some epochs and, consequently, the similarity of neurons. As a result, the topology search using cosine similarity might stop as well. To prevent this, we begin a random search when the classification accuracy on the validation set stops increasing. This phase is almost similar to phase 1, and they are different in the weight regrowing policy. In this phase, instead of using cosine similarity information, we add connections randomly to the network. In this way, we prevent early stopping of the topology search. Algorithm 1 summarizes this method.

3.4 Simultaneous Cosine Similarity-based and Random Topology Exploration (CTRE$_{\text{sim}}$)

To constantly exploit the cosine similarity information during training and avoid early stopping of topology exploration, we propose another method for obtaining a sparse neural network, named Simultaneous Cosine Similarity-based and Random Topology Exploration (CTRE$_{\text{sim}}$).

Prior to the training, we initialize a sparse neural network. After that, the training procedure starts with three steps in each epoch. The first two steps are similar to the CTRE$_{\text{seq}}$, which are (a) standard feed-forward and back-propagation, and (b) magnitude-based weight removal. However, in step (c), instead of relying solely on cosine similarity information or random addition, we combine both strategies. There are two reasons behind this choice: (1) As discussed in Section 3.3, as the training proceeds, the activation values become stable and might not change significantly after a while and, consequently, the similarity values. In CTRE$_{\text{seq}}$, we addressed this issue by switching completely to random search. However, the training speed might slow down if we rely only on the random search. (2) If we rely only on cosine similarity information, there is a possibility to add some connections based on the similarity of the neurons, which have been removed based on the magnitude in the weight removal step. It means that in these cases, the path between these pairs of similar neurons

does not contribute to the performance of the network. Therefore, we should not add such connections to the network. These are the potential limitations of CTRE_{seq}.

To address these limitations, CTRE_{sim} takes another approach to prevent adding the removed connections which have a high cosine similarity to the network, as follows. In step *c*, we add the connections with high similarities to the network; however, if some connections with high cosine similarity are earlier removed based on their magnitude in step *b*, we add random connections to the network. In other words, we split our budget between similarity-based and random exploration. More importantly, we let the network dynamically decide how much budget should be allocated to each exploration at each epoch. The benefits from this approach are twofold; we prevent early stopping of the topology search, and ales prevent re-adding connections that have shown to be unhelpful for the network's performance. Algorithm 2 summarizes this method.

4 Experiments and Results

In this section, we evaluate our proposed algorithms and compare them with several state-of-the-art algorithms for obtaining a sparse neural network. First, we describe the settings of the conducted experiments, including the hyperparameter values, implementation details, and datasets. Then, we compare them in terms of the classification accuracy on several datasets and networks with different sizes and sparsity levels.

4.1 Settings

This section gives a brief overview of the experiment settings, including hyperparameter values, implementation details, and datasets used for the evaluation of the methods.

4.1.1 Hyperparameters.

The network that we use to perform experiments is a 3-layer MLP as described in Section 3.1. The activation functions used for hidden and output layers are "Relu" and "Softmax", respectively, and the loss function used is "CrossEntropy". The values for most hyperparameters have been selected using a grid search over a limited number of values. The hyperparameter ζ has been set to 0.2. In Algorithm 1, $e_{early\ stop}$ has been set to 40. We train the network with Stochastic Gradient Decent (SGD) with momentum and L_2 regularizer. The momentum coefficient, the regularization coefficient, and learning rate are 0.9, 0.0001, and 0.01, respectively. All the experiments are performed using 500 training epochs. The datasets have been preprocessed using the Min-Max Scaler so that each feature is normalized between 0 and 1, except for Madelon, where we use standard scaler (each feature will have zero mean and unit variance). For the image datasets, data augmentation has not been performed unless it has been explicitly stated.

4.1.2 Implementation

We evaluate our proposed methods on eight datasets and compare the results with three state-of-the-art methods for obtaining sparse neural networks, including, RigL, SNIP, and SET. Besides, we measure the performance using a fully-connected MLP as the baseline method. We implemented our proposed method using Tensorflow [1]. The baseline of this

Table 1: Datasets characteristics.

Dataset	Dimensions	Type	Samples	Train	Test	Classes
Isolet	617	Speech	7737	6237	1560	26
Madelon	500	Artificial	2600	2000	600	2
MNIST	784	Image	70000	60000	10000	10
Fashion_MNIST	784	Image	70000	60000	10000	10
CIFAR10	3072	Image	60000	50000	10000	10
CIFAR100	3072	Image	60000	50000	10000	100
PCMAC	3289	Text	1943	1554	389	2
BASEHOCK	4862	Text	1993	1594	399	2

implementation is the RigL code from Github [2]. It also includes the implementation for SNIP, SET, and fully-connected MLP. This code uses a binary mask over weights to implement sparsity. In addition, we provide a purely sparse implementation that uses Scipy library sparse matrices. This code is developed from the sparse implementation of SET, which is available on Github[3]. For all the experiments, we use the Tensorflow implementation to have a fair comparison among methods. However, we provide the results using the sparse implementation in Appendix C. Most experiments were run on a CPU (Dell R730). For image datasets, we used a Tesla-P100 GPU. All the experiments were repeated with three random seeds. To ensure a fair comparison, for the sparse training methods (SET, RigL, and CTRE), the sparsity mask is updated at the end of each epoch, and drop fraction (ζ) and learning rate are constant during training.

4.1.3 Datasets

We conducted our experiments on eight benchmark datasets as follows:

- **Madelon** [21] is an artificial dataset with 20 informative features , and 480 noise features.
- **Isolet** [11] has been created with the spoken name of each letter of the English alphabet.
- **MNIST** [36] is a database of 28×28 images of handwritten digits.
- **Fashion_MNIST** [74] is a database of 28×28 images of Zalando's articles.
- **CIFAR10/100** [32] are two datasets of 32×32 colour images categorized in 10/100 classes.
- **PCMAC & BASEHOCK** [35] are two subsets of the 20 Newsgroups data.

More details about the datasets is presented in Table 1.

4.2 Performance Evaluation

In this experiment, we compare the methods in terms of classification accuracy on networks with varying sizes and sparsity levels. We consider three MLPs, each having three hidden layers with 100, 500, and 1000 hidden neurons, respectively. By changing the value of ε for each MLP, we study the effect of sparsity level on the performance of the methods. Table 2 summarizes the results of these experiments that are carried out on the five datasets, including tabular and image datasets that have different characteristics. We have also included the

[2] The implementation of RigL, SNIP, and SET is available at *https://github.com/google-research/rigl*.

[3] The pure sparse implementation of SET can be found on *https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks*.

Table 2: Classification accuracy (%) comparison among methods on networks with various sizes and sparsity levels. The density (%) and number of connections for each case is indicated in the table. Please note that N (total number of parameters of the network) is scaled by $\times 10^3$.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ε			ε			ε		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	Baseline (N)	54.3 ± 2.1 (120.0)			54.7 ± 1.2 (1000.0)			54.3 ± 2.1 (3000.0)		
	$D(\%)$ (N)	0.9 (1.1)	4.6 (5.5)	11.9 (14.3)	0.4 (3.5)	1.8 (17.5)	4.6 (45.5)	0.2 (6.5)	1.1 (32.5)	2.8 (84.5)
	SNIP	57.1 ± 4.7	53.8 ± 2.7	54.3 ± 2.8	58.1 ± 1.6	55.4 ± 4.3	58.8 ± 2.5	56.3 ± 4.9	58.2 ± 0.4	56.4 ± 1.7
	RigL	63.2 ± 0.7	60.2 ± 0.2	58.7 ± 2.7	50.0 ± 0.0	62.3 ± 0.5	60.9 ± 3.0	50.0 ± 0.0	63.3 ± 1.2	59.8 ± 0.2
	SET	61.2 ± 0.8	58.3 ± 1.5	58.0 ± 0.9	61.4 ± 1.1	58.2 ± 0.7	56.3 ± 1.7	60.4 ± 2.0	58.7 ± 0.8	59.6 ± 1.0
	CTRE_{seq}	73.1 ± 6.8	**62.0 ± 1.8**	59.9 ± 1.5	61.7 ± 2.5	**75.3 ± 2.1**	**62.2 ± 2.6**	62.1 ± 1.8	**74.2 ± 1.4**	59.9 ± 1.5
	CTRE_{sim}	**77.7 ± 1.9**	59.8 ± 1.5	**60.6 ± 2.1**	**80.0 ± 1.9**	74.8 ± 0.2	61.4 ± 2.1	**80.2 ± 0.3**	71.3 ± 1.0	**64.3 ± 0.3**
Isolet	Baseline (N)	94.4 ± 0.1 (143.4)			94.3 ± 0.0 (1117.7)			94.6 ± 0.4 (3234.0)		
	$D(\%)$ (N)	0.9 (1.2)	4.3 (6.2)	11.3 (16.2)	0.3 (3.6)	1.6 (18.2)	4.2 (47.4)	0.2 (6.6)	1.0 (33.2)	2.7 (86.4)
	SNIP	50.6 ± 6.3	90.3 ± 0.3	92.6 ± 1.2	53.9 ± 3.2	90.4 ± 0.9	91.7 ± 0.3	54.8 ± 4.3	88.2 ± 1.4	91.8 ± 0.3
	RigL	81.4 ± 0.5	89.8 ± 0.9	91.9 ± 0.4	3.5 ± 0.0	89.0 ± 1.1	91.4 ± 1.1	3.5 ± 0.0	87.0 ± 0.6	91.9 ± 0.5
	SET	**88.4 ± 1.9**	**93.7 ± 0.7**	**95.1 ± 0.2**	85.8 ± 0.7	**92.6 ± 0.6**	**94.6 ± 0.5**	73.3 ± 5.2	**92.6 ± 0.7**	**94.0 ± 0.6**
	CTRE_{seq}	86.7 ± 0.6	92.3 ± 1.4	94.6 ± 0.4	**89.3 ± 1.2**	91.6 ± 1.1	94.0 ± 0.4	87.2 ± 1.7	92.0 ± 0.6	93.6 ± 0.4
	CTRE_{sim}	85.4 ± 2.5	93.1 ± 0.1	93.9 ± 1.3	88.2 ± 0.9	92.1 ± 0.5	93.5 ± 1.1	**89.8 ± 1.0**	**92.6 ± 0.2**	92.9 ± 1.0
MNIST	Baseline (N)	97.9 ± 0.0 (176.8)			98.2 ± 0.1 (1284.0)			98.2 ± 0.1 (3568.0)		
	$D(\%)$ (N)	0.8 (1.4)	3.9 (7.0)	10.2 (18.1)	0.3 (3.8)	1.5 (19.0)	3.8 (49.3)	0.2 (6.8)	1.0 (34.0)	2.5 (88.3)
	SNIP	91.0 ± 0.5	96.2 ± 0.2	97.3 ± 0.0	93.6 ± 0.2	96.7 ± 0.2	97.3 ± 0.3	94.6 ± 0.3	96.6 ± 0.3	97.3 ± 0.1
	RigL	94.7 ± 0.6	96.4 ± 0.0	97.2 ± 0.1	96.0 ± 0.2	96.8 ± 0.1	96.9 ± 0.1	95.8 ± 0.2	96.6 ± 0.0	96.7 ± 0.1
	SET	95.4 ± 0.2	96.9 ± 0.1	97.5 ± 0.2	95.9 ± 0.2	97.4 ± 0.1	97.9 ± 0.1	95.9 ± 0.2	97.4 ± 0.1	97.8 ± 0.1
	CTRE_{seq}	**95.8 ± 0.1**	97.1 ± 0.2	**97.7 ± 0.1**	**97.0 ± 0.1**	**97.6 ± 0.1**	**98.0 ± 0.1**	**97.2 ± 0.1**	97.6 ± 0.2	97.8 ± 0.1
	CTRE_{sim}	95.6 ± 0.1	**97.4 ± 0.0**	**97.7 ± 0.1**	96.9 ± 0.1	97.5 ± 0.0	**98.0 ± 0.1**	97.1 ± 0.1	**97.7 ± 0.1**	**98.0 ± 0.1**
Fashion-MNIST	Baseline (N)	88.3 ± 0.2 (176.8)			89.8 ± 0.1 (1284.0)			90.1 ± 0.1 (3568.0)		
	$D(\%)$ (N)	0.8 (1.4)	3.9 (7.0)	10.2 (18.1)	0.3 (3.8)	1.5 (19.0)	3.8 (49.3)	0.2 (6.8)	1.0 (34.0)	2.5 (88.3)
	SNIP	81.1 ± 0.3	86.1 ± 0.2	87.2 ± 0.0	81.8 ± 0.9	86.5 ± 0.3	87.4 ± 0.0	81.5 ± 0.8	86.3 ± 0.1	87.4 ± 0.2
	RigL	85.0 ± 0.2	86.9 ± 0.2	87.1 ± 0.2	86.2 ± 0.3	86.3 ± 0.3	86.8 ± 0.1	86.4 ± 0.1	86.6 ± 0.4	86.7 ± 0.4
	SET	**85.9 ± 0.1**	**87.6 ± 0.0**	87.8 ± 0.1	86.2 ± 0.4	87.7 ± 0.2	88.0 ± 0.2	86.5 ± 0.1	87.6 ± 0.1	87.7 ± 0.2
	CTRE_{seq}	85.3 ± 0.4	87.3 ± 0.2	87.5 ± 0.1	86.8 ± 0.0	87.9 ± 0.4	88.0 ± 0.4	87.2 ± 0.1	**88.1 ± 0.2**	88.3 ± 0.2
	CTRE_{sim}	85.8 ± 0.2	87.4 ± 0.3	**87.9 ± 0.2**	**87.0 ± 0.2**	**88.2 ± 0.3**	**88.4 ± 0.1**	**87.4 ± 0.1**	87.8 ± 0.3	**88.4 ± 0.1**
CIFAR10	Baseline (N)	51.2 ± 0.5 (634.4)			53.2 ± 0.2 (3572.0)			55.2 ± 0.2 (8144.0)		
	$D(\%)$ (N)	0.6 (3.7)	2.9 (18.4)	7.5 (47.9)	0.2 (6.1)	0.9 (30.4)	2.2 (79.1)	0.1 (9.1)	0.6 (45.4)	1.4 (118.1)
	SNIP	36.6 ± 3.2	47.7 ± 0.6	49.1 ± 1.2	39.6 ± 0.4	49.5 ± 0.4	51.2 ± 0.7	41.2 ± 1.3	49.0 ± 0.8	51.0 ± 0.7
	RigL	46.5 ± 0.2	49.7 ± 0.8	50.1 ± 0.4	36.5 ± 18.8	50.2 ± 0.6	49.6 ± 0.7	22.9 ± 18.2	50.3 ± 0.3	50.4 ± 0.5
	SET	47.8 ± 0.0	49.9 ± 0.6	**50.6 ± 0.7**	49.3 ± 0.3	50.8 ± 0.4	51.5 ± 0.2	48.8 ± 0.4	50.7 ± 0.5	51.3 ± 0.1
	CTRE_{seq}	47.1 ± 0.3	**50.7 ± 0.5**	50.0 ± 0.4	50.8 ± 0.8	52.0 ± 0.3	52.5 ± 0.6	**51.7 ± 0.5**	**53.6 ± 0.4**	**54.2 ± 0.4**
	CTRE_{sim}	**48.5 ± 0.6**	50.4 ± 0.7	50.1 ± 0.2	**52.0 ± 0.4**	**53.2 ± 0.8**	**53.2 ± 0.4**	50.9 ± 0.7	52.6 ± 0.4	53.8 ± 0.4

density (as percentage) and the number of connections (divided by 10^3) for each network in this table. For training on each dataset, we allocate 10% of the training set to a validation set. During training, each MLP is trained on the new training set. At each epoch, we measure the performance on the validation set. Finally, Table 2 presents the results of each algorithm on an unseen test set and using the model that gives the highest validation accuracy during training. The learning curves regarding each case are presented in Appendix A; however, we present some interesting cases in Figure 2.

First, we analyze the performance of methods on the two tabular datasets. As can be seen in Table 2, on Madelon dataset, CTRE_{sim} is the best performer in most cases. Interestingly,

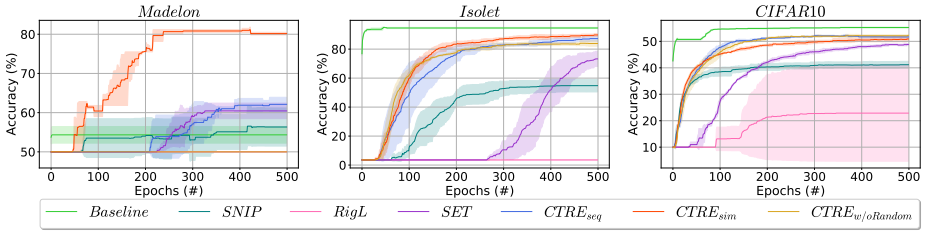

Fig. 2: Classification accuracy (%) comparison among methods on a highly large and sparse 3-layer MLP with a density lower than 0.22% ($n^l = 1000$, $\varepsilon = 1$).

the accuracy increases when the network becomes sparser. However, this can be explained intuitively; since the Madelon dataset contains many noise features ($> 95\%$), the higher the number of the connections is, the higher the risk for over-fitting the noise features will be. CTRE_{sim} can find the most important information paths in the network, which most likely start from the input neurons corresponding to the informative features. As a result, it can reach an accuracy of 80.2% with only 0.2% of total connections of the equivalent dense network ($n^l = 1000$), while the maximum accuracy achieved by other methods considered is 63.3% (RigL). On the second tabular dataset, Isolet, CTRE_{sim} and CTRE_{seq} are the best performers on the two most sparse models, 0.2% and 0.3% densities, respectively. In addition, in all the other cases, CTRE_{sim} and CTRE_{seq} are the second and third-best performers. In terms of learning speed, we can observe in Figure 2 that CTRE_{sim} can find a good topology much faster than other methods, which results in an increase in the accuracy within a short period after the training starts. From Figure 2, it can be seen that RigL fails to find an informative sub-network in these cases ($D < 0.3\%$). This indicates that gradient information might not be informative in highly sparse networks.

On the image dataset, CTRE_{sim} and CTRE_{seq} are the best and second-best performers in most of the cases considered. When the network size is small ($n^l = 100$), SET is the major competitor of CTRE. However, when the model size increases, CTRE outperforms SET. This indicates that the pure random weight addition policy in SET can perform well in networks with a higher density, while it is hard to find such sub-network randomly in high sparsity scenarios due to the very large search space. RigL also has a comparable performance to SET, except for very sparse models. As discussed in the previous paragraph, on a highly sparse network ($D < 0.3\%$), RigL has poor performance. Besides, as shown in Figure 2, SNIP starts with a steep increase in the accuracy due to the few iterations of training a dense network and thus, starting with good topology. However, as the training proceeds, this topology cannot achieve the same performance as other methods. Therefore, it indicates that dynamic weight update is an essential factor in the sparse training of neural networks.

These observations confirm that the cosine similarity is an informative criterion for adding weight in the network compared to random (SET) and gradient-based addition (RigL) in very sparse neural networks. CTRE can reach a better performance than state-of-the-art sparse training algorithms in terms of learning speed and accuracy when the network is highly sparse. Besides, by comparing the results with the dense network, it is clear that it is possible to reach a comparable performance to the dense network even with a network with 100 times fewer connections which is an excellent choice for low-resource devices on edge. We further

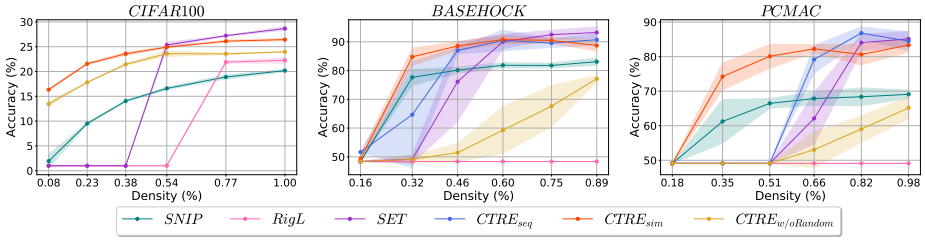

Fig. 3: Sparsity-accuracy trade-off on highly sparse neural networks on three datasets.

compare the computational cost of the algorithms in Appendix B and their learning speed in Appendix A.

4.3 Sparsity-Performance Trade-off Analysis in Highly Sparse MLPs

We carry out another experiment to study the trade-off between sparsity and accuracy on very high sparsity cases. We perform this experiment for two difficult classification tasks including, image classification on CIFAR100, which is considered as a more difficult dataset than the earlier considered image datasets, and text classification on PCMAC and BASESHOCK that are subsets of 20-newsgroup dataset; they have a high number of features and a low number of samples. This experiment uses a 3-layer MLP with 1000 and 3000 hidden neurons for text datasets and CIFAR100 dataset, respectively. We change the density value between 0 and 1 and compare our proposed approaches to SNIP, RigL, and SET (due to the close performance of CTRE_{sim} and CTRE_{seq} on earlier considered image datasets, on CIFAR100, we perform the experiments with CTRE_{sim}). We use data augmentation for CIFAR100. Also, as the network is considerably large on this dataset, we set the learning rate to 0.05 to speed up the training. The results are presented in Figure 3.

As shown in Figure 3, in highly sparse networks ($D < 0.5\%$), CTRE_{sim} outperforms other methods by a large gap. As discussed in Section 4.2, RigL performs poorly in these scenarios. SNIP outperforms SET and RigL at the very low densities while still has lower results than CTRE_{sim} in all cases. While SET outperforms other methods for larger density values on CIFAR100 and BASESHOCK, it performs poorly on a very sparse network. On text datasets, CTRE_{seq} has comparable performance to CTRE_{sim} and SET on higher densities, and it achieves the highest accuracy on PCMAC. Overall, we can observe that CTRE_{sim} has decent performance on these three datasets with a density value between 0.3% and 0.5%.

5 Discussion

In this section, we perform an in-depth analysis to understand the behavior of CTRE better. First, in Section 5.1, we perform two ablation studies to study the effectiveness of both random topology search and similarity importance metric in the performance of CTRE. In Section 5.2, we discuss why we have chosen magnitude over cosine similarity for the weight removal step. Finally, in Section 5.3, we discuss why the insensitivity of cosine similarity to the vector's magnitude is important in the performance of CTRE.

Table 3: Classification accuracy (%) comparison among Cosine similarity-based methods.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
			ε			ε			ε	
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	CTRE_{seq}	73.1±6.8	**62.0±1.8**	59.9±1.5	61.7±2.5	**75.3±2.1**	**62.2±2.6**	62.1±1.8	**74.2±1.4**	59.9±1.5
	CTRE_{sim}	77.7±1.9	59.8±1.5	**60.6±2.1**	**80.0±1.9**	74.8±0.2	61.4±2.1	**80.2±0.3**	71.3±1.0	**64.3±0.3**
	$\text{CTRE}_{\text{w/oRandom}}$	**79.6±2.0**	59.0±2.3	57.6±1.2	58.6±12.0	72.7±5.3	61.4±1.6	50.0±0.0	71.2±1.4	56.3±3.7
	$\text{CTRE}_{\text{sim/LTH}}$	61.3±3.4	58.0±2.6	56.9±1.7	53.2±0.7	59.6±1.4	58.2±1.7	51.1±0.2	58.9±1.6	58.3±1.2
Isolet	CTRE_{seq}	**86.7±0.6**	92.3±1.4	94.6±0.4	**89.3±1.2**	91.6±1.1	94.0±0.4	87.2±1.7	92.0±0.6	93.6±0.4
	CTRE_{sim}	85.4±2.5	93.1±0.1	93.9±1.3	88.2±0.9	**92.1±0.5**	93.5±1.1	**89.8±1.0**	**92.6±0.2**	92.9±1.0
	$\text{CTRE}_{\text{w/oRandom}}$	66.7±11.3	91.9±0.3	93.5±0.2	84.5±0.8	89.9±0.3	92.8±0.5	83.9±1.9	87.6±1.1	92.1±0.9
	$\text{CTRE}_{\text{sim/LTH}}$	81.1±1.3	**93.3±0.6**	**95.0±0.2**	66.3±4.9	91.7±0.6	**94.6±0.4**	61.2±11.8	91.8±1.1	**93.8±0.2**
MNIST	CTRE_{seq}	**95.8±0.1**	97.1±0.2	**97.7±0.1**	**97.0±0.1**	**97.6±0.1**	**98.0±0.1**	**97.2±0.1**	97.6±0.2	97.8±0.1
	CTRE_{sim}	95.6±0.1	**97.4±0.0**	**97.7±0.1**	96.9±0.1	97.5±0.0	**98.0±0.1**	97.1±0.1	**97.7±0.1**	**98.0±0.1**
	$\text{CTRE}_{\text{w/oRandom}}$	94.8±0.3	97.0±0.2	97.5±0.1	96.9±0.0	97.4±0.1	97.4±0.1	**97.2±0.3**	97.5±0.2	97.5±0.1
	$\text{CTRE}_{\text{sim/LTH}}$	94.6±0.2	96.9±0.1	97.4±0.1	94.2±0.1	97.3±0.2	97.4±0.0	93.9±0.0	97.1±0.0	97.5±0.1
Fashion-MNIST	CTRE_{seq}	85.3±0.4	87.3±0.2	87.5±0.1	86.8±0.0	87.9±0.4	88.0±0.4	87.2±0.1	**88.1±0.2**	88.3±0.2
	CTRE_{sim}	**85.8±0.2**	**87.4±0.3**	**87.9±0.2**	**87.0±0.2**	**88.2±0.3**	**88.4±0.1**	**87.4±0.1**	87.8±0.3	**88.4±0.1**
	$\text{CTRE}_{\text{w/oRandom}}$	83.9±0.4	86.9±0.3	86.9±0.1	86.2±0.3	87.9±0.2	88.0±0.2	86.8±0.3	87.7±0.4	88.2±0.2
	$\text{CTRE}_{\text{sim/LTH}}$	85.0±0.2	87.3±0.3	87.6±0.2	84.4±0.3	87.5±0.3	87.8±0.3	84.2±0.2	87.6±0.2	88.0±0.2
CIFAR10	CTRE_{seq}	47.1±0.3	**50.7±0.5**	50.0±0.4	50.8±0.8	52.0±0.3	52.5±0.6	51.7±0.5	**53.6±0.4**	**54.2±0.4**
	CTRE_{sim}	**48.5±0.6**	50.4±0.7	50.1±0.2	**52.0±0.4**	**53.2±0.8**	**53.2±0.4**	50.9±0.7	52.6±0.4	53.8±0.4
	$\text{CTRE}_{\text{w/oRandom}}$	45.4±0.2	49.4±0.3	49.9±0.4	50.9±0.3	52.2±0.6	52.7±0.4	**52.3±0.2**	53.3±0.5	53.5±0.4
	$\text{CTRE}_{\text{sim/LTH}}$	46.7±0.5	49.6±0.5	**50.4±0.3**	45.4±0.5	50.9±0.2	51.2±0.4	44.9±0.5	50.6±0.1	50.9±0.3

5.1 Ablation Study: Analysis of Topology Search Policies

This section presents and discusses the results of two ablation studies designed to understand better the effect of different topology search policies in CTRE. In the following, we describe each ablation experiment separately.

5.1.1 Ablation Study 1: Random Topology Search

The first ablation study aims to analyze the effect of random connection addition on the behavior of CTRE. Therefore, instead of using the similarity information and random search (simultaneously in CTRE_{sim} and sequentially in CTRE_{seq}), we only use the cosine similarity information at each epoch. We call this approach $\text{CTRE}_{w/oRandom}$ and repeat the experiments from Section 4.2. The detailed results are available in Table 3.

As can be seen in Table 3, in most cases considered $\text{CTRE}_{w/oRandom}$ has been outperformed by CTRE_{sim} and CTRE_{seq}. On the other hand, we can observe that on image datasets, $\text{CTRE}_{w/oRandom}$ has comparable performance to the other two methods; this indicates the effectiveness of similarity information on the image datasets. However, on tabular datasets, it performs poorly on high sparsity cases ($\varepsilon = 1$). Therefore, using only cosine information in these scenarios can cause the topology search to be stuck in a local minimum. This might have been originated by the early stopping of changes in the activation values, which leads to an early stop in topology search. CTRE_{seq} solves this by changing the weight update policy to random search. However, there is a risk of early switching to random search when the cosine information has not been fully exploited. Finally, by considering both random and cosine information in each epoch, the CTRE_{sim} algorithm will minimize the risk of staying in the local minimum or switching to a completely random search, both of which might slow the training process. In the context of network topology search, these components can also be

characterized as exploitation (local information based on the similarity between neurons) and exploration (random search). As a result, CTRE_{sim} can mitigate the limitations of CTRE_{seq} and find a performant sub-network by leveraging these two components, which outperform state-of-the-art algorithms.

5.1.2 Ablation Study 2: Cosine similarity-based Topology Search

To study the effectiveness of cosine similarity addition in the performance of CTRE, we design an experiment; in this experiment, we add connections in the reverse order of importance to the network. We expect that adding weights in this order would result in poor performance. We perform this experiment on CTRE_{sim}. Concretely, at each step, we add a number of weights with the lowest similarity among the corresponding neurons; if a weight with a very low similarity has been removed in the last weight removal step, we add a random connection instead. We call this method $\text{CTRE}_{\text{sim/LTH}}$ (LTH refers to low to high importance).

As can be seen in Table 3, $\text{CTRE}_{\text{sim/LTH}}$ has been outperformed by CTRE_{sim} and CTRE_{seq} in most of the cases considered. This shows that cosine similarity is a useful metric to detect the most important weights in the network. By comparing $\text{CTRE}_{\text{sim/LTH}}$ with SET (Table 2), it is clear that in most cases $\text{CTRE}_{\text{sim/LTH}}$ has a close or slightly worse accuracy than SET. Therefore, it can be inferred that $\text{CTRE}_{\text{sim/LTH}}$ is selecting non-informative weights, which can be similar to or worse than a random search. As a result, this can indicate the effectiveness of the introduced similarity metric (Equation 2) in finding a well-performing sparse neural network. It is worth noting that on the Isolet dataset, $\text{CTRE}_{\text{sim/LTH}}$ outperforms CTRE_{sim} and CTRE_{seq} in some cases, particularly in the networks with higher density. This is similar to the results of SET as well. Therefore, we can conclude that random search outperforms other methods on the Isolet dataset and low sparsity levels. However, it is not easy to find a highly sparse network using the random search policy.

5.2 Analysis of Weight Removal Policy

In this section, we aim to analyze the weight removal policy and further explain the reason behind choosing magnitude-based pruning over the cosine similarity (discussed in Section 3.2). In many previous studies, magnitude-based pruning has been commonly used as a criterion to remove unimportant weight from a neural network. We design an experiment to compare the performance of magnitude-based and cosine similarity-based pruning in neural networks.

In this experiment, we start with a trained network and gradually remove weights based on the magnitude and cosine similarity value (Using Equation 2) of the corresponding connection. We also consider random pruning as the baseline.

Settings. We perform this experiment using two networks: (1) A 3-layer dense MLP with 1000 neurons in each layer, and (2) A 3-layer sparse MLP with 1000 neurons in each layer that is trained using the SET approach [52] (3.8% density). The choice of SET instead of CTRE was made to avoid any biases on the cosine similarity weight removal, as CTRE uses cosine information to add weights. Both of these networks are trained on the MNIST dataset.

Weight Removal. We remove weights with two orders on each of the sparse and dense networks: least to most important and vice versa. We gradually remove weights; at each step, we remove 1% of the connections and measure the accuracy of the pruned network until no connection remains in the network.

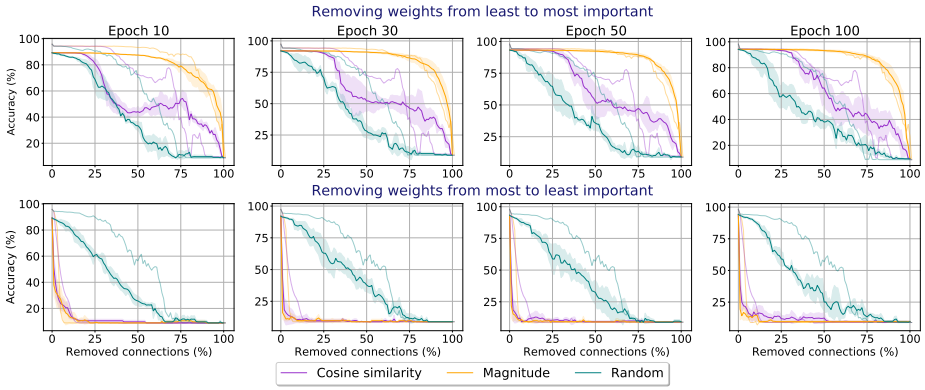

Fig. 4: Effect of weight removal using three criteria including magnitude, cosine similarity, and random, on the classification accuracy (%) at different epochs. The lines with higher transparency corresponds to the weight removal of the SET-MLP and the lines with lower transparency corresponds to the dense-MLP

Results. The results when the two networks are trained for 10, 30, 50, and 100 epochs are available in Figure 4. In this figure, the lines with higher transparency correspond to the weight removal of the SET-MLP, and the lines with lower transparency correspond to the dense-MLP. This experiment has been repeated with three seeds for each case.

As shown in Figure 4, when weights are removed from least to most important, magnitude-based pruning can order weights better than cosine similarity-based pruning. When the networks are trained for 100 epochs, by dropping the unimportant weights using magnitude, the major accuracy drop starts almost after removing 70% of the connections, while it happens after removing 30% for cosine similarity. This behavior exists in both the dense and the sparse networks. As expected, the drop for random removal happens from the beginning of the pruning procedure. In earlier epochs (10, 30, and 50), the drop in the accuracy happens earlier for both magnitude and cosine similarity.

It can be seen in Figure 4, by removing weights in the opposite order (from most to least important), the behavior of drop in the accuracy is almost similar for cosine similarity-based and magnitude-based pruning in SET-MLP, particularly in the earlier epochs. Therefore, both magnitude and cosine similarity can identify the most important connections in good order. However, this behavior is different in the dense network; magnitude-based pruning can better detect the most important weights. In the dense network, the drop in the accuracy for magnitude-based pruning happens earlier than cosine similarity pruning.

Conclusions. These observations can lead us to conclude that, firstly, the magnitude can be a good metric for weight removal in sparse training. Secondly, it can be inferred that cosine similarity can be a good metric for adding the most important connections in the weight addition phase in sparse neural networks in the absence of magnitude. As discussed earlier, the cosine similarity information of each connection is an informative criterion to detect the most important weights in a sparse neural network and has similar behavior to magnitude-based pruning in these scenarios. Therefore, in the absence of magnitude for non-existing connections in a sparse neural network (during weight addition), cosine similarity can be a useful criterion to detect the most important weights without requiring computing dense gradient information.

5.3 Magnitude Insensitivity: The Favorable Feature of Cosine Similarity in Noisy Environments

This section further discusses why cosine similarity has been chosen as a metric to determine the importance of non-existing connections. Specifically, we mainly focus on analyzing the importance of normalization in Equation 2 in the performance of the algorithm. While based on the Hebbian learning rule, the connection among a pair of neurons with high activations should be strengthened, we argue that in the search for a performant sparse neural network, the magnitude of the activations should be ignored.

Based on Hebb's rule (Section 2.2), the connection among the neurons with high activations receives higher synaptic updates. Therefore, if we evolve the topology using this rule (without any normalization) the importance of a non-existing connection should be determined by: $\left|\boldsymbol{A}^{l-1}_{:,p} \cdot \boldsymbol{A}^{l}_{:,q}\right|$. We evaluate the performance of this metric by replacing it with Equation 2 in CTRE_{sim} and CTRE_{seq}; we name these algorithms $\text{CTRE}_{\text{sim-Hebb}}$ and $\text{CTRE}_{\text{seq-Hebb}}$, respectively.

We evaluate these methods on the Madelon dataset. The reason behind choosing this dataset is due to its interesting properties; it contains 480 noisy features (out of the 500 features). Therefore, finding informative information paths through the network is considered to be a challenging task. The settings of this experiment are similar to Section 4.2; we measure the performance on networks with different sizes and sparsity levels. The results are presented in Table 4 and the accuracy during training is plotted in Figure 5. $\text{CTRE}_{\text{sim-Hebb}}$ and $\text{CTRE}_{\text{seq-Hebb}}$ have been outperformed by CTRE_{sim} and CTRE_{seq} in all cases considered. Particularly, we can observe that as the network becomes sparser, the gap between the performance of the pure Hebbian-based methods and the cosine similarity-based methods increases.

Table 4: Classification accuracy (%) comparison of Cosine similarity-based methods and pure Hebbian-based evolution, on the Madelon dataset.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
			ε			ε			ε	
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	CTRE_{seq}	73.1 ± 6.8	$\mathbf{62.0 \pm 1.8}$	59.9 ± 1.5	61.7 ± 2.5	$\mathbf{75.3 \pm 2.1}$	$\mathbf{62.2 \pm 2.6}$	62.1 ± 1.8	$\mathbf{74.2 \pm 1.4}$	59.9 ± 1.5
	CTRE_{sim}	$\mathbf{77.7 \pm 1.9}$	59.8 ± 1.5	$\mathbf{60.6 \pm 2.1}$	$\mathbf{80.0 \pm 1.9}$	74.8 ± 0.2	61.4 ± 2.1	$\mathbf{80.2 \pm 0.3}$	71.3 ± 1.0	$\mathbf{64.3 \pm 0.3}$
	$\text{CTRE}_{\text{seq-Hebb}}$	63.1 ± 1.0	58.8 ± 1.8	60.1 ± 0.9	60.8 ± 3.3	59.6 ± 0.3	59.4 ± 0.8	61.6 ± 2.9	56.8 ± 2.1	58.0 ± 0.6
	$\text{CTRE}_{\text{sim-Hebb}}$	59.6 ± 3.1	56.9 ± 7.9	60.1 ± 3.0	58.8 ± 4.2	62.7 ± 1.0	59.1 ± 1.1	60.2 ± 4.3	59.1 ± 2.1	59.4 ± 3.3

The poor performance of $\text{CTRE}_{\text{sim-Hebb}}$ and $\text{CTRE}_{\text{seq-Hebb}}$ on the Madelon dataset is resulted from their sensitivity to the magnitude of activation values. As Madelon contains many noisy features, some uninformative neurons likely receive a high activation value. Therefore, if we use only the activation magnitude to find the informative paths of information, the algorithm will be biased on the neurons with very high activation, which might not be informative. Therefore, it is likely to assign new connections to noisy features with high activation. This would cause the algorithm to be stuck in a local minimum which might be difficult to escape as these neurons continue to receive more and more connections at each epoch. Furthermore, as the networks become sparser, the informative features have a lower chance of receiving more connections (there are more noisy features compared to the informative ones). Therefore, in sparse networks, the gap between the performance of

these methods is much larger than in denser networks. Based on these observations, it can be concluded that the insensitivity of cosine similarity to the vector's magnitude helps CTRE to be more robust in noisy environments.

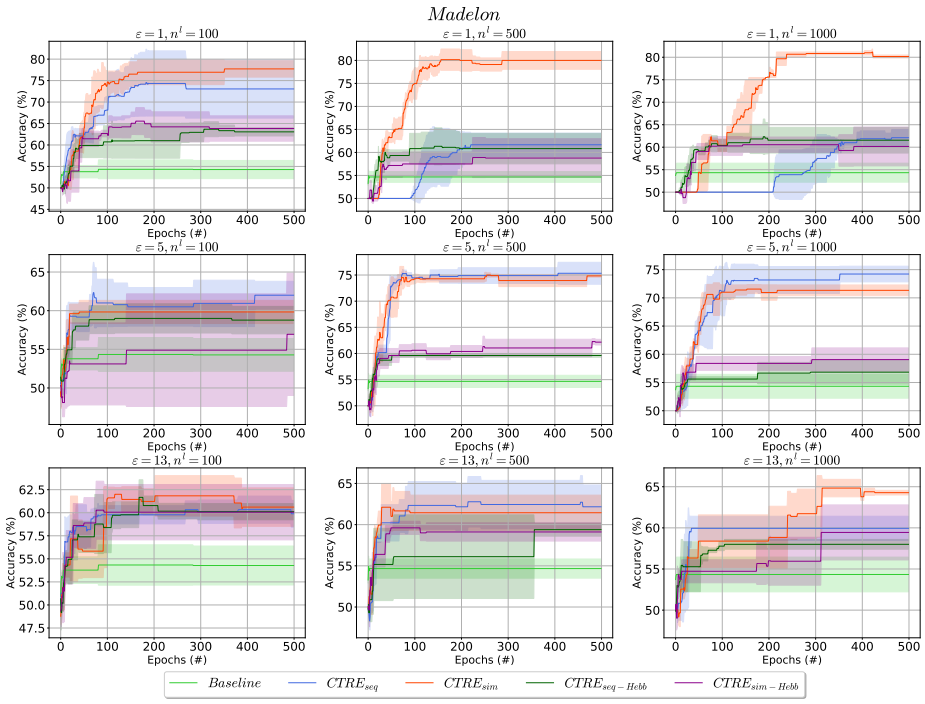

Fig. 5: Classification accuracy (%) comparison on Madelon for CTRE and pure Hebbian-based updates.

6 Conclusion and Broader Impacts

In this research, we introduced a new biologically plausible sparse training algorithm named CTRE. CTRE exploits both the similarity of neurons as an importance measure of the connections and random search, sequentially ($CTRE_{seq}$) or simultaneously ($CTRE_{sim}$), to explore a performant sparse topology. The findings of this study indicate that the cosine similarity between neurons' activations can help to evolve a sparse network in a purely sparse manner even in highly sparse scenarios, while most state-of-the-art methods may fail in these cases. In our view, by using the neurons' similarity to evolve the topology, our proposed approach can be an excellent initial step toward explainable sparse neural networks. Overall, due to the ability of CTRE to extract highly sparse neural networks, it can be a viable alternative for saving energy in both low-resource devices and data centers and pave the way to achieving environmentally friendly AI systems. Nevertheless, the trade-off between accuracy and sparsity, with CTRE deployed on real-world applications, should be considered carefully;

particularly, if any loss of accuracy may pose safety risks to the user, the sparsity level of the network needs to be analyzed with greater care.

An interesting future direction of this research is to extend CTRE to CNNs; driven by the decent performance of CTRE on image datasets, we believe that it has the potential to be extended to CNN architectures. However, in-depth theoretical analysis and systematic experiments are required to adapt this similarity metric to CNN architectures. This is due to the fact that CNNs require weight sharing, which does not exist in real neurons, and consequently, it is not straightforward to apply Hebbian learning directly [60]. There have been some efforts to make CNNs more biologically plausible [60, 4]. Therefore, applying CTRE to CNNs should be done with great care and theoretical analysis that we believe is in the scope of future works.

Declarations

- **Funding.** This project is partially funded by the NWO EDIC project.
- **Conflicts of interest/Competing interests.** Not Applicable.
- **Ethics approval.** Not Applicable.
- **Consent to participate.** Not Applicable.
- **Consent for publication.** Not Applicable.
- **Availability of data and material.** All the datasets used in this research are currently public. The references for all datasets have been cited in Section 4.1.3.
- **Code availability.** The implementation code is available on Github at *https://github.com/zahraatashgahi/CTRE*.
- **Authors' contributions.** J. Pieterse and D.C. Mocanu designed and developed a preliminary idea and proof of concept experiments. Z. Atashgahi designed and developed the CTRE algorithm, designed and carried out the empirical validation, and performed the analysis and interpretation of the results. S. Liu helped in performing the experiments and interpreting the results. D.C. Mocanu, R. Veldhuis and M. Pechenizkiy supervised the research. All authors provided critical feedback and helped shaped the research. Z. Atashgahi wrote the manuscript with input from all authors.

References

1. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL *https://www.tensorflow.org/*. Software available from tensorflow.org.
2. Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *International conference on machine learning*, pages 584–592. PMLR, 2014.

3. Zahra Atashgahi, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. *arXiv preprint arXiv:2012.00560*, 2020.
4. Sergey Bartunov, Adam Santoro, Blake A Richards, Luke Marris, Geoffrey E Hinton, and Timothy P Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9390–9400, 2018.
5. Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. URL *https://openreview.net/forum?id=BJ_wN01C-*.
6. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL *https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf*.
7. Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.
8. Pau de Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081*, 2020.
9. Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.
10. Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.
11. Mark Fanty and Ronald Cole. Spoken letter recognition. In *Advances in Neural Information Processing Systems*, pages 220–226, 1991.
12. Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
13. Karl Friston. Hierarchical models in the brain. *PLoS computational biology*, 4(11): e1000211, 2008.
14. Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.
15. Lukas Galke and Ansgar Scherp. Forget me not: A gentle reminder to mind the simple multi-layer perceptron baseline for text classification. *arXiv preprint arXiv:2109.03777*, 2021.
16. Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2018.
17. Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *arXiv preprint arXiv:2106.11959*, 2021.

18. Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.
19. AI High-Level Expert Group. Assessment list for trustworthy artificial intelligence (ALTAI) for self-assessment, 2020.
20. Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1387–1395, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.
21. Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.
22. Jiawei Han, Micheline Kamber, Jian Pei, et al. Getting to know your data. In *Data mining*, pages 39–82. Elsevier Amsterdam, Netherlands, 2012.
23. Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 1135–1143, 2015.
24. Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.
25. Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.
26. Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.
27. Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.
28. Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.
29. Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.
30. LIU Junjie, XU Zhe, SHI Runbin, Ray CC Cheung, and Hayden KH So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2019.
31. Jeremy Kepner and Ryan Robinett. Radix-net: Structured sparse matrices for deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 268–274. IEEE, 2019.
32. Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
33. Eduard Kuriscak, Petr Marsalek, Julius Stroffek, and Peter G Toth. Biological context of hebb learning in artificial neural networks, a review. *Neurocomputing*, 152:27–35, 2015.
34. Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning*

Research, pages 5544–5555. PMLR, 13–18 Jul 2020. URL *http://proceedings.mlr.press/v119/kusupati20a.html*.
35. Ken Lang. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pages 331–339. Elsevier, 1995.
36. Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.
37. Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.
38. Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL *https://openreview.net/forum?id=B1VZqjAcYX*.
39. Baoli Li and Liping Han. Distance weighted cosine similarity measure for text classification. In *International conference on intelligent data engineering and automated learning*, pages 611–618. Springer, 2013.
40. Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8018–8027, 2020.
41. Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3367–3375, 2015.
42. Congcong Liu and Huaming Wu. Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Processing*, 156:84–91, 2019. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2018.10.019. URL *https://www.sciencedirect.com/science/article/pii/S0165168418303517*.
43. Jia Liu, Maoguo Gong, and Qiguang Miao. Modeling hebb learning rule for unsupervised learning. In *IJCAI*, pages 2315–2321, 2017.
44. Shiwei Liu, Tim van der Lee, Anil Yaman, Zahra Atashgahi, Davide Ferrar, Ghada Sokar, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Topological insights into sparse neural networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) 2020.*, pages 2006–14085, 2020.
45. Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33(7):2589–2604, 2021.
46. Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. *arXiv preprint arXiv:2101.09048*, 2021.
47. Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. *arXiv preprint arXiv:2102.02887*, 2021.
48. Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018. URL *https://openreview.net/forum?id=H1Y8hhg0b*.
49. Chunjie Luo, Jianfeng Zhan, Xiaohe Xue, Lei Wang, Rui Ren, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. In *International Conference on Artificial Neural Networks*, pages 382–391. Springer, 2018.

50. Iacopo Masi, Yue Wu, Tal Hassner, and Prem Natarajan. Deep face recognition: A survey. In *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, pages 471–478. IEEE, 2018.
51. Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2-3):243–270, 2016.
52. Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.
53. Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H Nguyen, Madeleine Gibescu, Damien Ernst, and Zita A Vale. Sparse training theory for scalable and efficient agents. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 34–38, 2021.
54. Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.
55. Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. 2016.
56. Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
57. Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4646–4655. PMLR, 09–15 Jun 2019. URL *http://proceedings.mlr.press/v97/mostafa19a.html*.
58. Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL *https://openreview.net/forum?id=BygfghAcYX*.
59. Hieu V Nguyen and Li Bai. Cosine similarity metric learning for face verification. In *Asian conference on computer vision*, pages 709–720. Springer, 2010.
60. Roman Pogodin, Yash Mehta, Timothy P Lillicrap, and Peter E Latham. Towards biologically plausible convolutional networks. *arXiv preprint arXiv:2106.13031*, 2021.
61. Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.
62. Md Aamir Raihan and Tor M Aamodt. Sparse weight activation training. *arXiv preprint arXiv:2001.01969*, 2020.
63. Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11380–11390. Curran Associates, Inc., 2020. URL *https://proceedings.neurips.cc/paper/2020/file/83004190b1793d7aa15f8d0d49a13eba-Paper.pdf*.
64. Benjamin Scellier and Yoshua Bengio. Towards a biologically plausible backprop. *arXiv preprint arXiv:1602.05179*, 914, 2016.
65. Thomas Schumacher. Livewired neural networks: Making neurons that fire together wire together. *arXiv preprint arXiv:2105.08111*, 2021.

66. Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, 18(3):491–504, 2014.
67. Yi Sun, Xiaogang Wang, and Xiaoou Tang. Sparsifying neural network connections for face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4856–4864, 2016.
68. Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33, 2020.
69. Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.
70. Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning*, pages 6566–6575. PMLR, 2019.
71. Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.
72. Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2082–2090, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.
73. Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. *Information Sciences*, 307:39–52, 2015. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2015.02.024. URL *https://www.sciencedirect.com/science/article/pii/S0020025515001243*.
74. Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.
75. Jun Yang, Wenjing Xiao, Chun Jiang, M Shamim Hossain, Ghulam Muhammad, and Syed Umar Amin. Ai-powered green cloud and data center. *IEEE Access*, 7:4195–4203, 2018.
76. Mi Zhang, Faen Zhang, Nicholas D Lane, Yuanchao Shu, Xiao Zeng, Biyi Fang, Shen Yan, and Hui Xu. Deep learning in the era of edge computing: Challenges and opportunities. *Fog Computing: Theory and Practice*, 2020.
77. Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Appendix

A Performance Evaluation

In this appendix, we compare the performance of the algorithms in terms of the accuracy, the learning speed, and computational complexity. Particularly, we analyze the results obtained in Section 4.2 in the manuscript. We first introduce a new metric for comparing the learning speed of the methods. We also include the learning curves for the experiments of Section 4.2 in Figures 6, 7, 8, 9, and 10, which corresponds to Madelon, Isolet, MNIST, Fashion-MNIST, and CIFAR10, respectively. The characteristics of these datasets are presented in Table 1.

To compare the training speed, we define a metric that computes the fraction of the total training time required to reach a certain level of accuracy. We call this metric Training Delay (*TD*) and compute it as follows:

$$TD = \frac{\min_{acc_i \geqslant th \times acc_{max},\, i \in \{1,2,\ldots,\#\,epochs\}} i}{\#\,epochs}, \tag{3}$$

where acc_i is the test accuracy at epoch i, th is the threshold hyperparameter between 0 and 1, acc_{max} is the maximum accuracy achieved by the training methods for the model with n^l hidden neurons and sparsity level ε, and $\#\,epochs$ is the total number of training epochs. In other words, *TD* shows the trade-off between accuracy and learning speed. The lower *TD* is for a method, the faster it can be trained to reach a certain desired level of accuracy (determined by th); therefore, it has a better trade-off between accuracy and learning speed. We believe that minimizing this metric is crucial for low-resource devices where accuracy is not the only important aspect for evaluating the performance of the method. Instead, achieving a decent level of accuracy within a minimum number of training epochs is the primary concern.

For each network with different sizes and each training method, we measure *TD* on all datasets. We consider only the high sparsity case (when $\varepsilon = 1$) since when the network is dense, all the methods have very low *TD*, and the difference between them is negligible. However, when we are looking for a highly sparse sub-network, it takes longer for each method to find the well-performing sub-network, and the difference among the methods is more apparent. We set the threshold th to 0.9; therefore, we compute the training delay for reaching 0.9 of the maximum accuracy achieved on this model. The results are presented in Table 5. If a method cannot reach the $th \times acc_{max}$ within the total number of epochs (500 in these experiments), we keep the corresponding entry empty.

As can be seen in Table 5, CTRE_{sim} has the lowest training delay (*TD*) in 9 out of 15 cases considered. On Isolet and Madelon, some methods cannot reach the required level of accuracy (0.9 of the maximum accuracy) within the 500 training epochs. SNIP has the worst performance among these methods and cannot reach the required level of accuracy on Madelon, Isolet, and CIFAR10. RigL has a similar performance to SNIP; while it has comparable performance to other methods on Fashion-MNIST and MNIST, it has a poor performance on the other datasets. Finally, SET also has decent performance on Fashion-MNIST and MNIST. However, when the network is highly sparse and large ($\varepsilon = 1$ and $n^l > 100$) on the Isolet dataset, it does not have a good performance.

Table 5: Training delay (TD) (%) comparison among methods. Empty fields indicate that the method cannot reach the considered level of accuracy in 500 training epochs.

		Madelon			**Isolet**			**MNIST**			**Fashion-MNIST**			**CIFAR10**		
		n^l			n^l			n^l			n^l			n^l		
ε	**Method**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**
	SNIP	–	–	–	–	–	–	2.2	2.6	3.4	**1.4**	6.4	6.8	–	–	–
	RigL	–	–	–	86.8	–	–	3.0	4.0	6.4	2.6	**3.2**	**4.6**	22.0	–	–
1	SET	–	–	–	**40.4**	78.6	–	4.2	11.8	17.4	3.2	9.2	14.6	12.0	39.2	70.6
	CTRE_{seq}	24.2	–	–	46.6	46.0	49.2	2.0	2.4	3.2	4.4	14.0	10.6	23.6	15.6	**18.6**
	CTRE_{sim}	**15.0**	**17.2**	**34.4**	61.0	**34.0**	**33.4**	**1.8**	**2.2**	**2.4**	2.8	4.4	4.8	11.8	**14.2**	28.4
	$\text{CTRE}_{\text{w/oRandom}}$	17.6	–	–	–	52.4	49.0	2.2	2.8	3.4	3.6	5.4	7.4	**10.6**	15.2	22.2

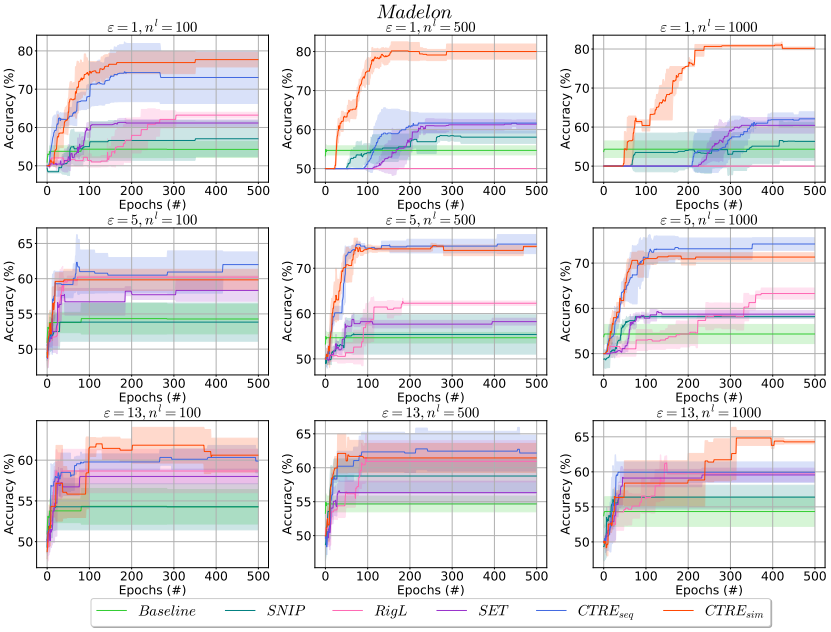

Fig. 6: Classification accuracy (%) results on Madelon.

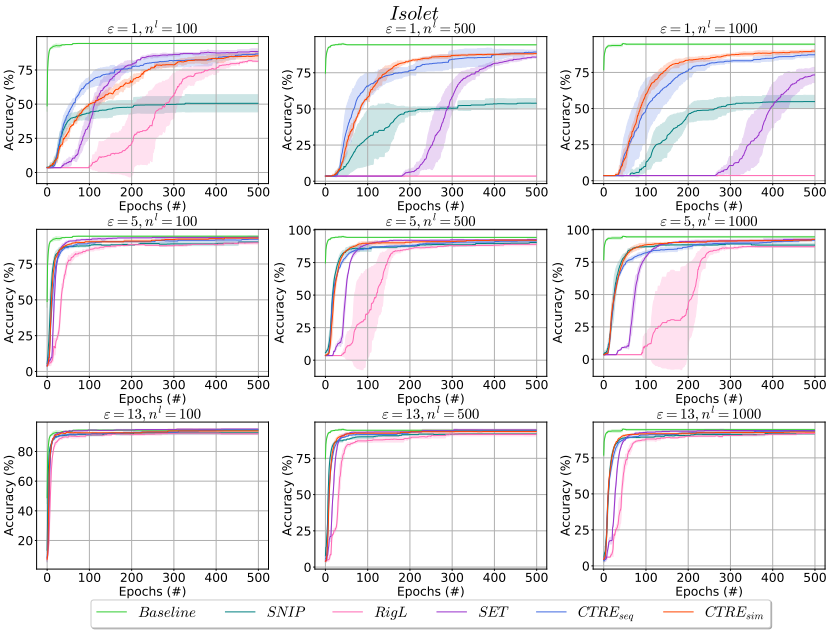

Fig. 7: Classification accuracy (%) results on Isolet.

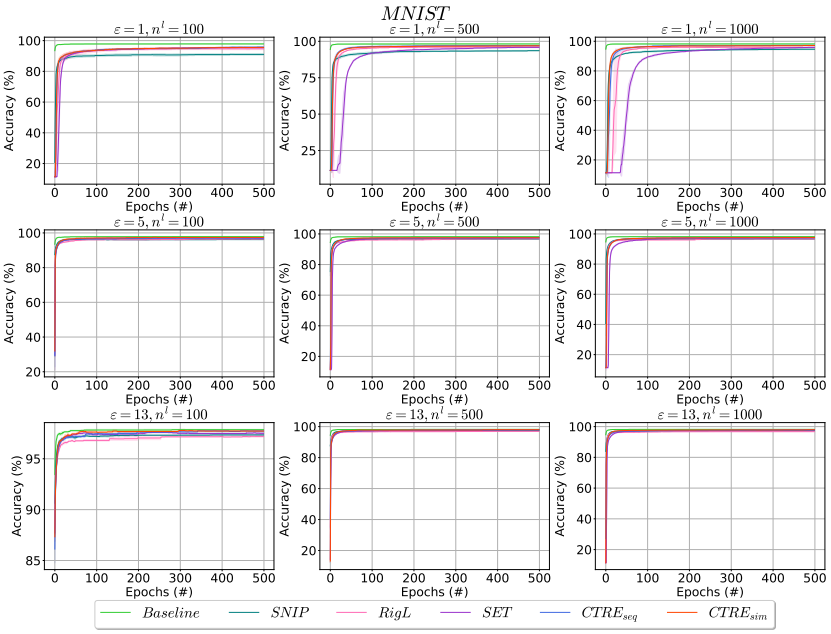

Fig. 8: Classification accuracy (%) results on MNIST.

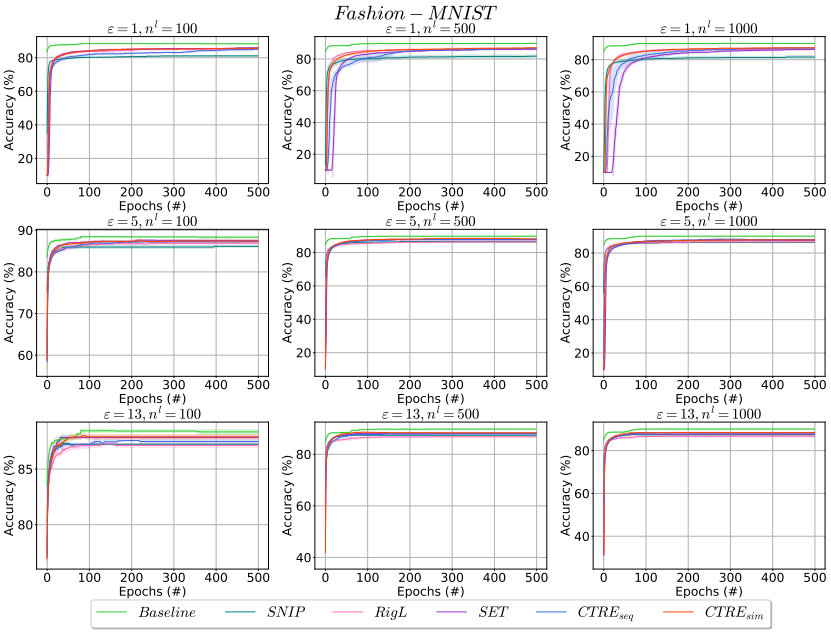

Fig. 9: Classification accuracy (%) results on Fashion-MNIST.

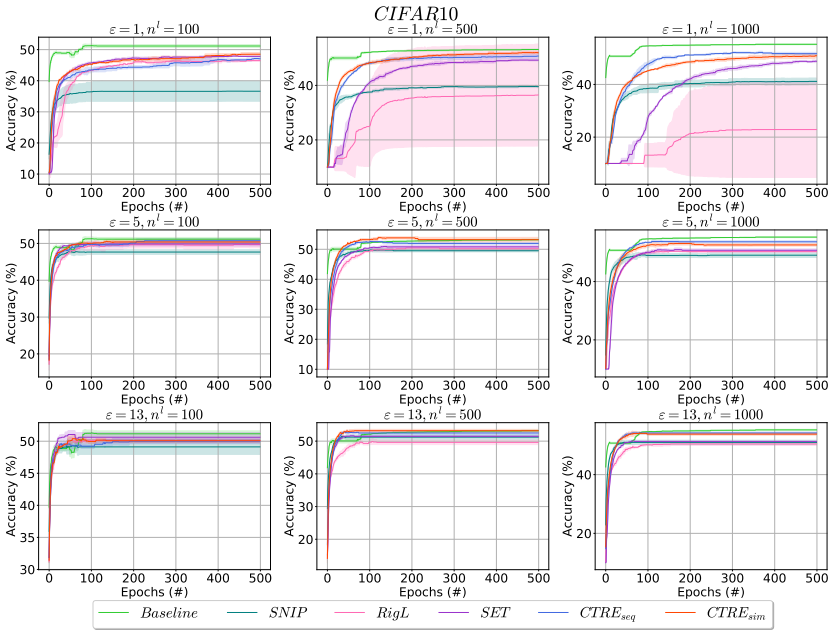

Fig. 10: Classification accuracy (%) results on CIFAR10.

B Computational Complexity

In this appendix, we compare the algorithms in terms of the computational complexity. While the computational cost during inference is equal for all methods (in the case of having the same sparsity level), the computational complexity during training is different.

We compare the computational complexity with the two closest sparse training algorithms to CTRE: SET and RigL. Our proposed methods require an extra cost of computing the cosine similarity matrix for the connections compared to SET. For each layer in each epoch, CTRE requires computing three dot products of size m (number of samples) for each connection in this layer to compute similarity matrix in Equation 2. Therefore, for each layer l, CTRE requires in the order of $\mathcal{O}(mN^l)$ extra computations at each epoch, where N^l is the number of parameters of layer l. However, this additional cost considerably improves the accuracy and the learning speed (discussed in Appendix A), particularly on tabular datasets and highly sparse neural networks. Therefore, depending on the application, the specialists should decide about the trade-off between accuracy and the computational cost when finding highly sparse neural networks. Compared to RigL, which requires computing occasional dense gradients, CTRE has the same order of complexity. This is because the order of computing gradients for back-propagation is also $\mathcal{O}(mN^l)$. However, CTRE outperforms RigL, especially in the high sparsity region.

To further decrease the computational cost of CTRE, we have tried to reduce the cost of cosine similarity computation by considering a proportion of the samples to compute the similarity matrix. We run CTRE_{sim} with half of the samples ($\text{CTRE}_{\text{sample2}}$) and a quarter of samples ($\text{CTRE}_{\text{sample4}}$) to compute the cosine

Table 6: Classification accuracy (%) of CTRE_{sim} with different number of training samples for computing the similarity matrices. Please note that N (total number of parameters of the network) is scaled by $\times 10^3$.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ε			ε			ε		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	Baseline (N)	54.3 ± 2.1 (120.0)			54.7 ± 1.2 (1000.0)			54.3 ± 2.1 (3000.0)		
	$D(\%)$ (N)	0.9 (1.1)	4.6 (5.5)	11.9 (14.3)	0.4 (3.5)	1.8 (17.5)	4.6 (45.5)	0.2 (6.5)	1.1 (32.5)	2.8 (84.5)
	CTRE_{sim}	**77.7 ± 1.9**	59.8 ± 1.5	60.6 ± 2.1	**80.0 ± 1.9**	**74.8 ± 0.2**	**61.4 ± 2.1**	**80.2 ± 0.3**	**71.3 ± 1.0**	**64.3 ± 0.3**
	$\text{CTRE}_{\text{sample2}}$	72.8 ± 1.4	**60.8 ± 1.1**	**61.1 ± 0.8**	76.3 ± 3.1	65.6 ± 3.4	60.4 ± 2.3	76.3 ± 1.8	69.4 ± 3.2	60.8 ± 3.0
	$\text{CTRE}_{\text{sample4}}$	67.4 ± 1.0	60.6 ± 0.6	58.6 ± 2.1	64.4 ± 2.4	59.7 ± 1.8	61.0 ± 0.6	68.8 ± 2.1	59.5 ± 1.2	58.8 ± 2.8
Isolet	Baseline (N)	94.4 ± 0.1 (143.4)			94.3 ± 0.0 (1117.7)			94.6 ± 0.4 (3234.0)		
	$D(\%)$ (N)	0.9 (1.2)	4.3 (6.2)	11.3 (16.2)	0.3 (3.6)	1.6 (18.2)	4.2 (47.4)	0.2 (6.6)	1.0 (33.2)	2.7 (86.4)
	CTRE_{sim}	**85.4 ± 2.5**	**93.1 ± 0.1**	93.9 ± 1.3	88.2 ± 0.9	92.1 ± 0.5	93.5 ± 1.1	**89.8 ± 1.0**	**92.6 ± 0.2**	92.9 ± 1.0
	$\text{CTRE}_{\text{sample2}}$	81.9 ± 4.2	92.8 ± 0.3	94.7 ± 0.3	**88.4 ± 0.8**	**92.6 ± 0.4**	**94.0 ± 1.1**	88.7 ± 0.5	92.4 ± 0.7	**94.0 ± 0.2**
	$\text{CTRE}_{\text{sample4}}$	85.0 ± 1.3	93.0 ± 0.8	**94.9 ± 0.5**	87.9 ± 0.5	91.7 ± 1.1	93.8 ± 0.1	89.4 ± 1.0	92.5 ± 0.2	93.8 ± 0.5
MNIST	Baseline (N)	97.9 ± 0.0 (176.8)			98.2 ± 0.1 (1284.0)			98.2 ± 0.1 (3568.0)		
	$D(\%)$ (N)	0.8 (1.4)	3.9 (7.0)	10.2 (18.1)	0.3 (3.8)	1.5 (19.0)	3.8 (49.3)	0.2 (6.8)	1.0 (34.0)	2.5 (88.3)
	CTRE_{sim}	95.6 ± 0.1	**97.4 ± 0.0**	**97.7 ± 0.1**	**96.9 ± 0.1**	97.5 ± 0.0	**98.0 ± 0.1**	**97.1 ± 0.1**	97.7 ± 0.1	**98.0 ± 0.1**
	$\text{CTRE}_{\text{sample2}}$	**95.7 ± 0.0**	97.0 ± 0.1	**97.7 ± 0.0**	96.3 ± 0.0	**97.8 ± 0.0**	97.8 ± 0.0	96.2 ± 0.0	**97.9 ± 0.1**	97.8 ± 0.1
	$\text{CTRE}_{\text{sample4}}$	95.1 ± 0.2	97.3 ± 0.1	97.6 ± 0.2	96.1 ± 0.2	97.7 ± 0.1	97.8 ± 0.1	96.6 ± 0.1	97.7 ± 0.1	97.8 ± 0.0
Fashion-MNIST	Baseline (N)	88.3 ± 0.2 (176.8)			89.8 ± 0.1 (1284.0)			90.1 ± 0.1 (3568.0)		
	$D(\%)$ (N)	0.8 (1.4)	3.9 (7.0)	10.2 (18.1)	0.3 (3.8)	1.5 (19.0)	3.8 (49.3)	0.2 (6.8)	1.0 (34.0)	2.5 (88.3)
	CTRE_{sim}	**85.8 ± 0.2**	87.4 ± 0.3	87.9 ± 0.2	87.0 ± 0.2	**88.2 ± 0.3**	88.4 ± 0.1	**87.4 ± 0.1**	87.8 ± 0.3	**88.4 ± 0.1**
	$\text{CTRE}_{\text{sample2}}$	85.7 ± 0.1	87.4 ± 0.1	87.7 ± 0.3	86.3 ± 0.7	87.9 ± 0.1	88.1 ± 0.2	86.3 ± 0.2	**88.3 ± 0.0**	88.0 ± 0.2
	$\text{CTRE}_{\text{sample4}}$	85.6 ± 0.4	**87.6 ± 0.1**	**88.0 ± 0.1**	**87.2 ± 0.1**	87.5 ± 0.1	**88.6 ± 0.1**	86.6 ± 0.3	88.0 ± 0.1	**88.4 ± 0.2**
CIFAR10	Baseline (N)	51.2 ± 0.5 (634.4)			53.2 ± 0.2 (3572.0)			55.2 ± 0.2 (8144.0)		
	$D(\%)$ (N)	0.6 (3.7)	2.9 (18.4)	7.5 (47.9)	0.2 (6.1)	0.9 (30.4)	2.2 (79.1)	0.1 (9.1)	0.6 (45.4)	1.4 (118.1)
	CTRE_{sim}	**48.5 ± 0.6**	50.4 ± 0.7	50.1 ± 0.2	**52.0 ± 0.4**	**53.2 ± 0.8**	**53.2 ± 0.4**	50.9 ± 0.7	52.6 ± 0.4	**53.8 ± 0.4**
	$\text{CTRE}_{\text{sample2}}$	47.8 ± 0.1	49.4 ± 0.4	50.9 ± 0.4	50.9 ± 0.3	52.8 ± 0.4	52.9 ± 0.4	50.4 ± 0.5	**53.9 ± 0.1**	53.4 ± 0.2
	$\text{CTRE}_{\text{sample4}}$	47.8 ± 0.1	**50.8 ± 0.2**	**51.1 ± 0.4**	**52.0 ± 0.1**	52.7 ± 0.3	52.9 ± 0.6	**51.6 ± 0.5**	53.6 ± 0.3	53.6 ± 0.5

similarity matrix. The results can be observed in Table 6. It is clear that even with half of the samples, CTRE can still achieve a close performance as the original method on all datasets except Madelon. On the Madelon dataset, CTRE_{sim} outperforms $\text{CTRE}_{\text{sample2}}$ and $\text{CTRE}_{\text{sample4}}$. The reason for this different behaviour is that Madelon is a highly noisy dataset and has a low number of samples compared to the other datasets (Table 1). Therefore, CTRE would be more sensitive to decreasing the number of samples in this case. Based on these observations, it can be concluded that in datasets with a high number of samples, only a fraction of samples can be used to compute the similarity matrix. In this way, we would be able to decrease the computational cost without affecting the performance.

Further studies can be performed to decrease the computational cost of deriving the similarity matrix. In short, CTRE is a first step in finding highly sparse neural networks using neuron characteristics and can be further explored in future works.

C Performance Evaluation Using Pure Sparse Implementation

In this appendix, we present the results using the pure sparse implementation. This code is developed from the sparse implementation of SET[4]. While the other training methods for obtaining sparse neural networks mostly use a binary mask over weights to simulate sparsity, this code is implemented in a purely sparse manner using Cython and SciPy sparse matrices. We have implemented our proposed method using this sparse implementation and repeated the experiments from Section 4.2 in the manuscript. The results are summarized in Table 7.

Table 7: Classification accuracy (%) comparison using pure sparse implementation. Please note that N (total number of parameters of the network) is scaled by $\times 10^3$.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ε			ε			ε		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	Baseline (N)	54.3 ± 2.1 (120.0)			54.7 ± 1.2 (1000.0)			54.3 ± 2.1 (3000.0)		
	$D(\%)$ (N)	0.9 (1.1)	4.6 (5.5)	11.9 (14.3)	0.4 (3.5)	1.8 (17.5)	4.6 (45.5)	0.2 (6.5)	1.1 (32.5)	2.8 (84.5)
	SET	58.7 ± 2.0	60.4 ± 3.3	58.1 ± 1.5	65.1 ± 2.2	59.6 ± 5.1	61.8 ± 1.0	61.8 ± 1.9	62.1 ± 4.7	61.6 ± 3.9
	$CTRE_{seq}$	85.3 ± 0.3	75.1 ± 0.6	67.1 ± 1.4	**87.2 ± 1.2**	82.1 ± 2.0	75.3 ± 1.0	87.2 ± 0.2	**86.6 ± 0.7**	75.7 ± 2.7
	$CTRE_{sim}$	82.9 ± 1.0	73.9 ± 2.3	66.8 ± 2.7	82.5 ± 1.1	79.4 ± 1.0	74.9 ± 1.2	82.9 ± 1.1	80.4 ± 1.6	74.6 ± 0.6
	$CTRE_{w/oRandom}$	**86.4 ± 1.2**	**75.6 ± 1.2**	**68.5 ± 0.6**	**87.2 ± 1.2**	**82.8 ± 1.1**	**77.0 ± 0.6**	**88.2 ± 0.6**	84.5 ± 0.6	**77.9 ± 1.4**
Isolet	Baseline (N)	94.4 ± 0.1 (143.4)			94.3 ± 0.0 (1117.7)			94.6 ± 0.4 (3234.0)		
	$D(\%)$ (N)	0.9 (1.2)	4.3 (6.2)	11.3 (16.2)	0.3 (3.6)	1.6 (18.2)	4.2 (47.4)	0.2 (6.6)	1.0 (33.2)	2.7 (86.4)
	SET	**87.6 ± 1.0**	**94.1 ± 0.1**	94.5 ± 0.4	86.2 ± 1.2	93.7 ± 0.7	94.4 ± 0.1	86.1 ± 0.5	94.1 ± 0.1	94.7 ± 0.1
	$CTRE_{seq}$	83.1 ± 1.2	93.7 ± 0.4	**94.6 ± 0.7**	**91.1 ± 0.3**	**94.2 ± 0.3**	**94.5 ± 0.4**	**90.9 ± 1.1**	**94.3 ± 0.4**	**94.8 ± 0.3**
	$CTRE_{sim}$	85.6 ± 0.6	93.2 ± 0.7	**94.6 ± 0.6**	88.6 ± 1.3	93.7 ± 0.4	94.4 ± 0.2	88.8 ± 0.6	93.5 ± 0.1	94.0 ± 0.1
	$CTRE_{w/oRandom}$	81.1 ± 1.1	92.9 ± 1.6	**94.6 ± 0.2**	89.1 ± 0.5	93.8 ± 0.1	93.9 ± 0.3	90.6 ± 0.9	93.9 ± 0.4	93.4 ± 0.7
MNIST	Baseline (N)	97.9 ± 0.0 (176.8)			98.2 ± 0.1 (1284.0)			98.2 ± 0.1 (3568.0)		
	$D(\%)$ (N)	0.8 (1.4)	3.9 (7.0)	10.2 (18.1)	0.3 (3.8)	1.5 (19.0)	3.8 (49.3)	0.2 (6.8)	1.0 (34.0)	2.5 (88.3)
	SET	95.0 ± 0.2	97.5 ± 0.1	97.8 ± 0.1	94.9 ± 0.2	97.5 ± 0.0	97.8 ± 0.1	95.2 ± 0.2	97.3 ± 0.1	97.6 ± 0.0
	$CTRE_{seq}$	**95.7 ± 0.2**	**97.8 ± 0.1**	**97.9 ± 0.0**	97.3 ± 0.2	**97.9 ± 0.2**	**98.0 ± 0.0**	**97.6 ± 0.0**	**97.9 ± 0.1**	**97.9 ± 0.1**
	$CTRE_{sim}$	95.4 ± 0.1	97.6 ± 0.0	97.7 ± 0.0	96.4 ± 0.6	97.4 ± 0.1	97.8 ± 0.0	96.7 ± 0.6	97.2 ± 0.0	97.7 ± 0.1
	$CTRE_{w/oRandom}$	95.5 ± 0.1	97.4 ± 0.0	97.7 ± 0.1	**97.4 ± 0.1**	97.7 ± 0.1	97.5 ± 0.2	**97.6 ± 0.1**	97.8 ± 0.0	97.8 ± 0.2
Fashion-MNIST	Baseline (N)	88.3 ± 0.2 (176.8)			89.8 ± 0.1 (1284.0)			90.1 ± 0.1 (3568.0)		
	$D(\%)$ (N)	0.8 (1.4)	3.9 (7.0)	10.2 (18.1)	0.3 (3.8)	1.5 (19.0)	3.8 (49.3)	0.2 (6.8)	1.0 (34.0)	2.5 (88.3)
	SET	**85.6 ± 0.1**	87.7 ± 0.1	**88.5 ± 0.1**	85.2 ± 0.2	87.9 ± 0.1	88.4 ± 0.3	85.2 ± 0.2	87.8 ± 0.0	88.5 ± 0.2
	$CTRE_{seq}$	85.4 ± 0.4	**88.0 ± 0.1**	88.4 ± 0.0	**86.8 ± 0.1**	**88.5 ± 0.2**	**88.6 ± 0.1**	**87.1 ± 0.4**	**88.6 ± 0.2**	**88.7 ± 0.2**
	$CTRE_{sim}$	84.9 ± 0.8	87.4 ± 0.3	88.1 ± 0.1	85.7 ± 0.6	87.7 ± 0.1	88.2 ± 0.1	85.9 ± 0.7	87.5 ± 0.1	88.4 ± 0.2
	$CTRE_{w/oRandom}$	83.7 ± 0.2	87.3 ± 0.2	87.7 ± 0.2	85.9 ± 0.2	87.9 ± 0.3	88.0 ± 0.1	86.5 ± 0.4	88.0 ± 0.2	87.9 ± 0.0
CIFAR10	Baseline (N)	51.2 ± 0.5 (634.4)			53.2 ± 0.2 (3572.0)			55.2 ± 0.2 (8144.0)		
	$D(\%)$ (N)	0.6 (3.7)	2.9 (18.4)	7.5 (47.9)	0.2 (6.1)	0.9 (30.4)	2.2 (79.1)	0.1 (9.1)	0.6 (45.4)	1.4 (118.1)
	SET	48.5 ± 0.4	52.7 ± 0.5	**54.0 ± 0.4**	47.7 ± 0.4	53.3 ± 0.3	54.3 ± 0.4	46.2 ± 0.2	52.8 ± 0.2	54.5 ± 0.7
	$CTRE_{seq}$	48.5 ± 0.5	**53.3 ± 0.3**	53.0 ± 0.6	**52.2 ± 0.3**	**55.7 ± 0.6**	**55.6 ± 0.3**	**54.2 ± 0.3**	**55.8 ± 0.1**	**56.2 ± 0.1**
	$CTRE_{sim}$	**48.6 ± 0.0**	51.0 ± 0.0	51.3 ± 0.0	50.1 ± 0.6	55.4 ± 0.1	54.3 ± 0.2	50.2 ± 0.9	**55.8 ± 0.4**	55.8 ± 0.0
	$CTRE_{w/oRandom}$	45.5 ± 0.5	49.6 ± 0.3	49.0 ± 0.4	51.2 ± 0.3	53.5 ± 0.3	53.0 ± 0.1	53.8 ± 0.4	53.6 ± 0.0	54.7 ± 0.1

As can be seen in Table 7, the results are subtly different from Table 2 in Section 4.2. This difference arise from some small differences in the implementation. One of the main differences is that in Section 4.2, Tensoflow library is used for implementing the neural network; however, this implementation uses Numpy, Scipy, and Cython to perform sparse matrix operations. Another difference is the weight initialization

[4] The pure sparse implementation of SET can be found on *https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks*.

policy. While in the experiments of Section 4.2, weights are initialized using a uniform distribution, in the sparse implementation weights are initialized using a normal distribution which seems more beneficial to this implementation. Overall, in most cases, the results in Table 7 are higher than the results in Table 2.