A brain-inspired algorithm for training highly sparse neural networks

Zahra Atashgahi1 · Joost Pieterse2 · Shiwei Liu2 · Decebal Constantin Mocanu1,2 · Raymond Veldhuis1 · Mykola Pechenizkiy2,3

Received: 17 October 2021 / Revised: 10 May 2022 / Accepted: 2 July 2022
© The Author(s) 2022

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.

Keywords Deep Learning · Sparse neural networks · Sparse training

Editors: Krzysztof Dembczynski and Emilie Devijver.

Zahra Atashgahi
z.atashgahi@utwente.nl

1 Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500AE Enschede, The Netherlands
2 Faculty of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands
3 Faculty of Information Technology, University of Jyväskylä, 40014 Jyväskylä, Finland

Published online: 08 November 2022
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 (Graves et al., 2013), image processing (Liang & Hu, 2015; Masi et al., 2018), and natural language processing (NLP) (Brown et al., 2020). It has been shown in Hestness et al. (2017) that the performance of deep neural networks scales with model size and dataset size, and generalization benefits from over-parameterization (Neyshabur et al., 2019). 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 (Zhang et al., 2020). 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 (Yang et al., 2018). However, a trustworthy AI system should function in the most environmentally friendly way possible during development, and deployment (Group, 2020). 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 (Zhang et al., 2020).

Sparse neural networks have been considered as an effective solution to address these challenges (Hoeffler et al., 2021; Mocanu et al., 2021). 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 (Frankle & Carbin, 2018; Atashgahi et al., 2022), while having much fewer parameters. It has been shown that biological brains, especially the human brain, enjoy sparse connections among neurons (Friston, 2008). 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 Mocanu et al. (2021), in dense-to-sparse training, the pruning phase can be done after training (Frankle & Carbin, 2018; Han et al., 2015; LeCun et al., 1990), simultaneous to training (Louizos et al., 2018), or one-shot prior to training (Lee et al., 2019). 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 Mocanu et al. (2016), 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 (Kepner & Robinett, 2019; Mocanu et al., 2016, 2021)), or might dynamically change, by removing and re-adding weights (known as dynamic sparse connectivity (Mocanu et al., 2018; Bellec et al., 2018)). By optimizing the topology along with the weights during the training, dynamic sparse training algorithms outperform the static ones (Mocanu et al., 2018). As discussed in Mocanu et al. (2018), 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

---

1 https://github.com/zahraatashgahi/CTRE.
training methods use magnitude as a pruning criterion, weight regrowing approaches are of different types, including random (Mocanu et al., 2018; Mostafa & Wang, 2019) and gradient-based regrowth (Evci et al., 2020; Jayakumar et al., 2020). As shown in Liu et al. (2021c), 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 (Dettmers & Zettlemoyer, 2019). 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 Sect. 4.2, we demonstrate that some gradient-based sparse training algorithms might fail in a highly sparse neural network.

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” (Hebb, 2005), 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$_{sim}$, Fig. 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 Sect. 3.2.
- Our proposed algorithms outperform state-of-the-art sparse training algorithms in highly sparse neural networks.

![Schematic of the proposed approach (CTRE$_{sim}$).](image)

Fig. 1 Schematic of the proposed approach (CTRE$_{sim}$). At each epoch, after feed-forward and back-propagation, a fraction $\zeta$ of the weights with the smallest magnitude is dropped (red connections). Then, similarity matrices $Sim^1$ and $Sim^2$ are computed using Eq. 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).
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 (Popov et al., 2019). 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 (Gorishniy et al., 2021). 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 (Galke & Scherp, 2021; Tolstikhin et al., 2021). In addition, in Jouppi et al. (2017), 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 LeCun et al. (1990), Hassibi and Stork (1993), where authors use hessian matrix information to prune a trained dense network. More recently, in Han et al. (2015); Frankle and Carbin (2018), authors use magnitude to remove unimportant connections. Other metrics, such as gradient (Liu & Wu, 2019), Taylor expansion (Molchanov et al., 2016, 2019), and low-rank decomposition (Li et al., 2020; Wang et al., 2019a), 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 (Gale et al., 2019; Junjie et al., 2019; Kusupati et al., 2020). Various criteria can be used for pruning, such as magnitude (Guo et al., 2016; Zhu & Gupta, 2017), L0 regularization (Louizos et al., 2018; Savarese et al., 2020), group Lasso regularization (Wen et al., 2016), and variational dropout (Molchanov et al., 2017). (3) *Pruning before training*: The first study to apply pruning prior to training was done by Lee et al. (2019), 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 (Wang et al., 2019b), connection sensitivity after pruning (de Jorge et al., 2020), and Synaptic Flow (Tanaka et al., 2020).
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 (Kepner & Robinett, 2019; Mocanu et al., 2016)) or dynamic (dynamic sparse training (DST) (Bellec et al., 2018; Mocanu et al., 2018)). 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) (Mocanu et al., 2018) 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 Mostafa and Wang (2019), 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 (Atashgahi et al., 2022; Gordon et al., 2018; Liu et al., 2020, 2021a, b, c). (2) Gradient information: A group of works have tried to exploit gradient information to speed up the training process in DST (Raihan & Aamodt, 2020). Dettmers and Zettlemoyer (2019) 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) (Evci et al., 2020) addressed the high computational cost by using infrequent gradient information. However, it still requires the computational cost for computing the periodic dense gradients. Jayakumar et al. (2020) tried to further improve RigL by using the gradient for only a subset of non-existing weights. In Dai et al. (2019), 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 (Hebb, 2005) 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 \( \Delta w_{ij} \) is the change in synaptic weight \( w_{ij} \) between two neurons \( p_i \) (presynaptic) and \( q_j \) (postsynaptic) in two consecutive layers, and \( \eta \) is the learning rate. While some previous works have adapted Hebb’s rule to some machine learning tasks, Liu et al. (2017), Scellier and Bengio (2016) 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 (Kuriscak et al., 2015). In Arora et al. (2014), authors have incorporated the Hebbian learning theory to train a newly introduced neural network. In Sun et al. (2016), 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 Dai et al. (2019), 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 (Xia et al., 2015). 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 (Han et al., 2012). Due to its simplicity and efficiency, the cosine similarity is a widely used metric in machine learning and pattern recognition field (Xia et al., 2015). Cosine Similarity has proven to be an effective tool also in neural networks. In Luo et al. (2018), 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 Nguyen and Bai (2010), 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 $X$ and target output $y$, a dense neural network is trained to minimize $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)})$, where $m$ is the number of training samples, $L$ is the loss function, $f$ is a neural network parametrized by $\theta$, $f(x^{(i)}; \theta)$ is the predicted output for input $x^{(i)}$, and $y^{(i)}$ is the true label. $\theta \in \mathbb{R}^N$ is consisted of parameters of each layer $l \in \{1, 2, ..., H\}$ of the network as $\theta^l \in \mathbb{R}^{N^l}$, 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 $\theta^l$, and discards $s^l$ fraction of parameters of each layer $\theta^l$ (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 $\theta$. We aim to train this network to minimize the loss on the training set as follows:

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

where $||\theta||_0$ 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
\[ P(\theta'_i) = \frac{e^{\theta'_i} + e^{\theta'_i}}{e^{\theta'_i} + e^{\theta'_i}}, \text{ } i \in \{1, 2, ..., N_l\}, \] where \( e \in \mathbb{R}^+ \) denotes the hyperparameter that controls the sparsity level. The lower the value of \( e \) is, the sparser the network would be. In other words, by increasing \( e \), the probability of \( P(\theta'_i) \) would be higher which results in more connections and a denser network. 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 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:

\[
Sim^l_{p,q} = \frac{A^l_{\cdot p} \cdot A^l_{\cdot q}}{|A^l_{\cdot p}| \cdot |A^l_{\cdot q}|},
\]

where \( Sim^l_{p,q} \) is the similarity matrix between neurons in two successive layers \( l-1 \) and \( l \). \( A^l_{\cdot p} \) and \( A^l_{\cdot q} \in \mathbb{R}^n \) 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 (Schumacher, 2021). It has been discussed in Schumacher (2021) 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 (Sect. 2.2), the importance of a connection in our method is also determined by multiplying the activations of its corresponding neurons, albeit normalized; in Eq. 2, \( A^l_{\cdot p} \) is the presynaptic activation and \( A^l_{\cdot q} \) is the postsynaptic activation. If the activations of two connected neurons agree, by computing the dot product of activations, both Eq. 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 Sect. 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<sub>seq</sub>)

Our first proposed algorithm, Sequential Cosine Similarity-based and Random Topology Exploration (CTRE<sub>seq</sub>) 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 $s$, 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 of connections with the lowest magnitude in each layer is removed. In Sect. 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, we measure the similarity of two neurons as formulated in Eq. 2. In each layer, we add connections (as many connections as the removed connections in this layer) 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 $e_{\text{early\,stop}}$ epochs ($e_{\text{early\,stop}}$ is a hyperparameter of CTRE<sub>seq</sub>). 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$_{sim}$)

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$_{sim}$).

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$_{seq}$, 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 Sect. 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 CTREseq, 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 CTREseq.

To address these limitations, CTREsim 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 also 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 Sect. 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 $\zeta$ has been set to 0.2. In Algorithm 1, $\epsilon_{\text{early stop}}$ 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 Comparison

We compare the results with three state-of-the-art methods for obtaining sparse neural networks, including, SNIP, RigL, and SET.

- **SNIP** [Lee et al. (2019)](https://www.cs.toronto.edu/~mlane/). Single-shot network pruning (SNIP) is a dense-to-sparse sparsification algorithm that prunes the network prior to initialization based on connection sensitivity. It calculates this metric after a few iteration of dense training. After pruning, SNIP starts the training with the sparse neural network.

- **RigL** [Evci et al. (2020)](https://github.com/google-research/rigl). The rigged lottery (RigL) is a sparse-to-sparse algorithm for obtaining a sparse neural network that uses the gradient information as the weight addition criteria.

- **SET** [Mocanu et al. (2018)](https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks). Sparse evolutionary training (SET) is a sparse-to-sparse training algorithm that uses random weight addition for updating the topology.

Besides, we measure the classification performance of a fully-connected MLP as the baseline method.

4.1.3 Implementation

We evaluate our proposed methods and the considered baselines on eight datasets. We implemented our proposed method using Tensorflow [Abadi et al., 2015](https://www.tensorflow.org). The baseline of this implementation is the RigL code from Github. 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. 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. The only exception is the experiments from Sect. 4.2 where we run 15 random seeds to analyze the statistical significance of the obtained results with respect to the considered algorithms (Sect. 4.2.1). 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 ($\zeta$) and learning rate are constant during training.

4.1.4 Datasets

We conducted our experiments on eight benchmark datasets as follows:

- **Madelon** [Guyon et al. (2008)](https://www.cs.toronto.edu/~mlane) is an artificial dataset with 20 informative features, and 480 noise features.

- **Isolet** Fanty and Cole (1991) has been created with the spoken name of each letter of the English alphabet.

---

2 The implementation of RigL, SNIP, and SET is available at [https://github.com/google-research/rigl](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](https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks).
- MNIST LeCun (1998) is a database of 28 × 28 images of handwritten digits.
- Fashion_MNIST Xiao et al. (2017) is a database of 28 × 28 images of Zalando’s articles.
- CIFAR10/100 Krizhevsky et al. (2009) are two datasets of 32×32 colour images categorized in 10/100 classes.
- PCMAC & BASEHOCK Lang (1995) 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 $\varepsilon$ 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 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 Fig. 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, 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 78.8% with only 0.3% of total connections of the equivalent dense network ($n_l = 1000$), while the maximum accuracy achieved by other methods considered is 61.9% (SET). On the second tabular dataset, Isolet, CTRE$_{sim}$ is the best performers on two very sparse models, including 0.4% ($n_l = 500$) and 0.3% ($n_l = 1000$) densities. 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 Fig. 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 Fig. 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 Fig. 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 compare the computational cost of the algorithms in Appendix B and their learning speed in Appendix A.

4.2.1 Statistical significance analysis

In this section, we analyze the statistical significance of the results obtained by CTRE compared to the other algorithms. To measure this, we perform Kolmogorov-Smirnov test (KS-test). The null hypothesis is that the two independent results/samples are drawn from the same continuous distribution. If the p-value is very small (p-value $< 0.05$), it suggests that the difference between the two sets of results is significant and the hypothesis is rejected. Otherwise, the obtained results are close together and the hypothesis is true.

We perform KS-test between the results obtained by CTRE (for simplicity, we consider maximum results of $CTRE_{seq}$ and $CTRE_{sim}$) and the other considered algorithms for the experiments in Table 2. The results of the KS-test is summarized in Table 3. In this table, Reject shows that the results are sufficiently distinct and True means that the obtained results are close together. The * sign in Table 3 shows that an algorithm has achieved the maximum accuracy in the corresponding experiment. Finally, the entries colored as red shows an experiment where a compared method obtains a close result to CTRE while having lower mean accuracy.

From Table 3, we can observe that in majority of the experiments, CTRE obtains higher mean accuracy than the other methods while being statistically different from them. The only dataset where the results in most cases are close is the Fashion-MNIST dataset where SET has comparable results to CTRE on this dataset. In addition, in high sparsity regime
| Dataset      | Method             | $n^I = 100$                          | $n^I = 500$                          | $n^I = 1000$                         |
|-------------|--------------------|--------------------------------------|--------------------------------------|--------------------------------------|
|             | $\epsilon$        | 1  5  13                             | 1  5  13                             | 1  5  13                             |
| Madelon     | Baseline ($N(\times10^3)$) | 54.9 ± 1.0 (70.2)                    | 54.9 ± 1.0 (751.0)                   | 54.9 ± 1.0 (2502.0)                  |
|             | $D(\%)$ ($N(\times10^3)$) | 1.6 (1.1)                            | 7.8 (5.5)                            | 20.4 (14.3)                          |
|             | SNIP               | 56.5 ± 3.5                           | 56.1 ± 2.9                           | 54.1 ± 2.6                           |
|             | RigL               | 60.4 ± 3.1                           | 59.7 ± 1.8                           | 59.3 ± 2.2                           |
|             | SET                | 60.4 ± 2.1                           | 57.8 ± 3.2                           | 58.1 ± 1.7                           |
|             | CTRE$_{seq}$       | **82.2 ± 2.4**                       | 72.5 ± 2.0                           | 63.9 ± 1.2                           |
|             | CTRE$_{sim}$       | 81.6 ± 1.3                           | **73.0 ± 1.6**                       | **65.6 ± 3.0**                       |
| Isolet      | Baseline ($N(\times10^3)$) | 94.4 ± 0.1 (84.3)                    | 94.3 ± 0.0 (821.5)                   | 94.6 ± 0.4 (2643.0)                  |
|             | $D(\%)$ ($N(\times10^3)$) | 1.5 (1.2)                            | 7.4 (6.2)                            | 19.2 (16.2)                          |
|             | SNIP               | 48.9 ± 5.8                           | 90.1 ± 1.1                           | 92.8 ± 0.7                           |
|             | RigL               | 66.0 ± 28.6                          | 90.0 ± 0.8                           | 92.3 ± 0.7                           |
|             | SET                | **89.1 ± 1.2**                       | **94.2 ± 0.7**                       | **94.9 ± 0.4**                       |
|             | CTRE$_{seq}$       | 86.7 ± 1.8                           | 92.4 ± 1.1                           | 94.3 ± 0.5                           |
|             | CTRE$_{sim}$       | 87.5 ± 0.8                           | 93.4 ± 0.7                           | 94.3 ± 0.8                           |
| MNIST       | Baseline ($N(\times10^3)$) | 97.9 ± 0.0 (99.4)                    | 98.2 ± 0.1 (897.0)                   | 98.2 ± 0.1 (2794.0)                  |
|             | $D(\%)$ ($N(\times10^3)$) | 1.4 (1.4)                            | 7.0 (7.0)                            | 18.2 (18.1)                          |
|             | SNIP               | 91.1 ± 0.5                           | 96.3 ± 0.2                           | 97.2 ± 0.1                           |
|             | RigL               | 94.5 ± 0.3                           | 96.4 ± 0.2                           | 97.2 ± 0.2                           |
|             | SET                | 95.6 ± 0.3                           | 97.1 ± 0.1                           | 97.6 ± 0.1                           |
|             | CTRE$_{seq}$       | **95.7 ± 0.2**                       | **97.3 ± 0.2**                       | **97.7 ± 0.1**                       |
|             | CTRE$_{sim}$       | 95.5 ± 0.2                           | **97.3 ± 0.1**                       | **97.8 ± 0.1**                       |
Table 2 continued

| Dataset   | Method          | $n' = 100$ | $n' = 500$ | $n' = 1000$ |
|-----------|----------------|-----------|------------|-------------|
|           |                | $\epsilon$ | $\epsilon$ | $\epsilon$ |
| Fashion- MNIST | Baseline ($N \times 10^3$) | 88.3 ± 0.2 (99.4) | 89.8 ± 0.1 (897.0) | 90.1 ± 0.1 (2794.0) |
|           | $D$ (%) ($N \times 10^3$) | 1.4 (1.4) | 0.4 (3.8) | 0.2 (6.8) |
|           | SNIP | 79.5 ± 2.3 | 81.1 ± 1.2 | 82.2 ± 1.1 |
|           | RigL | 84.8 ± 0.2 | 86.1 ± 0.2 | 86.1 ± 0.3 |
|           | SET | 85.8 ± 0.3 | 86.5 ± 0.2 | 86.3 ± 0.2 |
|           | CTREseq | 85.8 ± 0.5 | 87.1 ± 0.4 | 87.3 ± 0.2 |
|           | CTREsim | 85.8 ± 0.3 | 88.0 ± 0.2 | 88.3 ± 0.2 |
| CIFAR10 | Baseline ($N \times 10^3$) | 51.2 ± 0.5 (328.2) | 53.2 ± 0.2 (2041.0) | 55.2 ± 0.2 (5082.0) |
|           | $D$ (%) ($N \times 10^3$) | 1.1 (3.7) | 0.3 (6.1) | 0.2 (9.1) |
|           | SNIP | 35.8 ± 3.8 | 39.4 ± 1.8 | 40.2 ± 1.4 |
|           | RigL | 46.0 ± 0.6 | 46.4 ± 9.8 | 36.4 ± 18.7 |
|           | SET | 48.3 ± 0.5 | 49.4 ± 0.3 | 48.8 ± 0.5 |
|           | CTREseq | 47.9 ± 0.8 | 50.1 ± 0.5 | 50.6 ± 0.5 |
|           | CTREsim | 48.2 ± 0.4 | 50.3 ± 0.3 | 50.7 ± 0.5 |
| CIFAR100 | Baseline ($N \times 10^3$) | 23.1 ± 0.1 (337.2) | 23.1 ± 0.6 (2086.0) | 23.1 ± 0.6 (5172.0) |
|           | $D$ (%) ($N \times 10^3$) | 1.1 (3.8) | 0.3 (6.2) | 0.2 (9.2) |
|           | SNIP | 5.9 ± 0.6 | 6.6 ± 0.8 | 6.2 ± 0.8 |
|           | RigL | 7.4 ± 4.4 | 1.0 ± 0.0 | 1.0 ± 0.0 |
|           | SET | 14.7 ± 0.4 | 14.3 ± 1.5 | 1.0 ± 0.0 |
|           | CTREseq | 12.7 ± 0.7 | 21.8 ± 0.5 | 21.4 ± 0.4 |
|           | CTREsim | 13.8 ± 0.4 | 21.9 ± 0.4 | 24.6 ± 0.4 |

Bold font indicates the best performer among the methods in the corresponding setting.

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 $\times 10^3$. 
and large network size \((n' = 1000, \varepsilon = 1)\), CTRE achieves the highest accuracy among the methods while being significantly distinct from them. Overall, Table 3 indicates that CTRE is a well performing algorithm in terms of the classification accuracy that achieves significantly different results from the other methods.

### 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\textsubscript{sim} and CTRE\textsubscript{seq} on earlier considered image datasets, on CIFAR100, we perform the experiments with CTRE\textsubscript{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 Fig. 3.

As shown in Fig. 3, in highly sparse networks \((D < 0.5\%)\), CTRE\textsubscript{sim} outperforms other methods by a large gap. As discussed in Sect. 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\textsubscript{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\textsubscript{seq} has comparable performance to CTRE\textsubscript{sim} and SET on higher densities, and it achieves the highest accuracy on PCMAC. Overall, we can observe that CTRE\textsubscript{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 Sect. 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 Sect. 5.2, we discuss why we have chosen magnitude over cosine similarity for the weight removal step. In Sect. 5.3, we discuss why the insensitivity of cosine similarity to the vector’s
Table 3: Statistical significance of the results

| Dataset       | 
|---------------|
|               |
| Dataset       | Method | $n' = 100$ | $n' = 500$ | $n' = 1000$ |
|               | $\epsilon$ | 1 | 5 | 13 | $\epsilon$ | 1 | 5 | 13 | $\epsilon$ | 1 | 5 | 13 |
| Madelon       | SNIP    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | RigL    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | SET     | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | CTRE    | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     |
| Isolet        | SNIP    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | RigL    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | SET     | Reject* | Reject* | Reject* | Reject* | Reject* | Reject* | Reject* | Reject* | Reject* | Reject* | Reject* |
|               | CTRE    | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     |
| MNIST         | SNIP    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | RigL    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | SET     | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  |
|               | CTRE    | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     |
| Fashion-MNIST | SNIP    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | RigL    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | SET     | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  |
|               | CTRE    | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     |
| CIFAR 10      | SNIP    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | RigL    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | SET     | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  |
|               | CTRE    | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     |
| CIFAR 100     | SNIP    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | RigL    | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject | Reject |
|               | SET     | Reject* | Reject | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  | True*  |
|               | CTRE    | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     | _*     |
| Dataset | Method | $n' = 100$ | $n' = 500$ | $n' = 1000$ |
|---------|--------|-----------|-----------|-------------|
|         |        | $\epsilon$ | $\epsilon$ | $\epsilon$ |
| CTRE    | –      | –*        | –*        | –*          |

Each entry show the result of KS-test among the results of CTRE and the compared methods for a specific network size and sparsity level. *Reject* shows that the results are distinct, and *True* indicates that the results are close together.
magnitude is important in the performance of CTRE. Finally, we discuss the convergence of CTRE in Sect. 5.4.

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 CTRE\(_{w/o\text{Random}}\) and repeat the experiments from Sect. 4.2. The detailed results are available in Table 4.

As can be seen in Table 4, in most cases considered, CTRE\(_{w/o\text{Random}}\) has been outperformed by CTRE_sim and CTRE_seq. On the other hand, we can observe that on image datasets, CTRE\(_{w/o\text{Random}}\) 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 CTRE_sim/LTH (LTH refers to low to high importance).

As can be seen in Table 4, CTRE_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 CTRE_sim/LTH with SET (Table 2), it is clear that in most cases CTRE_sim/LTH has a close or slightly worse accuracy.
than SET. Therefore, it can be inferred that CTRE\textsubscript{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 (Eq. 2) in finding a well-performing sparse neural network. It is worth noting that on the Isolet dataset, CTRE\textsubscript{sim/LTH} outperforms CTRE\textsubscript{sim} and CTRE\textsubscript{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 Sect. 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 Eq. 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 (Mocanu et al., 2018) (3.2\% 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.

**Results** The results when the two networks are trained for 10, 30, 50, and 100 epochs are available in Fig. 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 Fig. 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

![Fig. 3](image-url)
Table 4: Classification accuracy (%) comparison among Cosine similarity-based methods

| Dataset    | Method         | \( n' = 100 \) | \( n' = 500 \) | \( n' = 1000 \) |
|------------|----------------|-----------------|----------------|-----------------|
|            | \( \epsilon \) | \( \epsilon \)  | \( \epsilon \) | \( \epsilon \)  |
| Madelon    | CTRE\(_{seq}\) | 82.2 ± 2.4      | 72.5 ± 2.0     | 71.8 ± 2.0      | 61.1 ± 1.9      | 83.9 ± 2.0 | 76.5 ± 1.9 |
|            | CTRE\(_{w/o}\) | 81.6 ± 1.3      | 73.0 ± 1.6     | 73.0 ± 1.5      | 78.8 ± 2.2      | 78.5 ± 1.0 | 74.6 ± 1.4 |
|            | CTRE\(_{w/o\text{Random}}\) | 79.6 ± 2.0    | 59.0 ± 2.3     | 58.6 ± 12.0     | 50.0 ± 0.0      | 71.2 ± 1.4 | 56.3 ± 3.7 |
|            | CTRE\(_{w/o\text{LTH}}\) | 61.3 ± 3.4    | 58.0 ± 2.6     | 53.2 ± 0.7      | 51.1 ± 0.2      | 58.4 ± 1.9 | 58.3 ± 1.2 |
| Isolet     | CTRE\(_{seq}\) | 86.7 ± 1.8      | 92.4 ± 1.1     | 94.3 ± 0.5      | 86.6 ± 1.6      | 91.5 ± 1.0 | 93.7 ± 0.5 |
|            | CTRE\(_{w/o}\) | 87.5 ± 0.8      | 93.4 ± 0.7     | 93.7 ± 0.8      | 88.3 ± 0.7      | 91.3 ± 1.3 | 93.1 ± 0.6 |
|            | CTRE\(_{w/o\text{Random}}\) | 66.7 ± 11.3   | 91.9 ± 0.3     | 93.5 ± 0.2      | 92.8 ± 0.5      | 93.9 ± 1.9 | 92.1 ± 0.9 |
|            | CTRE\(_{w/o\text{LTH}}\) | 81.1 ± 1.3    | 93.3 ± 0.6     | 95.0 ± 0.2      | 94.6 ± 0.4      | 91.8 ± 1.1 | 93.8 ± 0.2 |
| MNIST      | CTRE\(_{seq}\) | 95.7 ± 0.2      | 97.3 ± 0.2     | 97.6 ± 0.2      | 97.6 ± 0.2      | 97.3 ± 0.1 | 97.7 ± 0.1 | 97.8 ± 0.1 |
|            | CTRE\(_{w/o}\) | 95.5 ± 0.2      | 97.3 ± 0.1     | 97.8 ± 0.1      | 96.4 ± 0.2      | 96.8 ± 0.2 | 97.7 ± 0.1 | 97.9 ± 0.1 |
|            | CTRE\(_{w/o\text{Random}}\) | 94.8 ± 0.3     | 97.0 ± 0.2     | 97.5 ± 0.1      | 96.9 ± 0.0      | 97.4 ± 0.1 | 97.4 ± 0.1 | 97.9 ± 0.1 |
|            | CTRE\(_{w/o\text{LTH}}\) | 94.6 ± 0.2     | 96.9 ± 0.1     | 97.4 ± 0.1      | 94.2 ± 0.1      | 97.3 ± 0.2 | 97.4 ± 0.0 | 97.5 ± 0.1 |
| Fashion-MNIST | CTRE\(_{seq}\) | 85.8 ± 0.5      | 87.5 ± 0.3     | 87.4 ± 0.3      | 87.1 ± 0.4      | 87.3 ± 0.2 | 88.0 ± 0.3 | 88.3 ± 0.2 |
|            | CTRE\(_{w/o}\) | 85.8 ± 0.3      | 87.5 ± 0.3     | 88.0 ± 0.2      | 86.4 ± 0.5      | 88.1 ± 0.3 | 88.3 ± 0.2 | 88.3 ± 0.2 |
|            | CTRE\(_{w/o\text{Random}}\) | 83.9 ± 0.4    | 86.9 ± 0.3     | 87.8 ± 0.3      | 86.4 ± 0.3      | 87.9 ± 0.3 | 88.4 ± 0.2 | 88.4 ± 0.2 |
|            | CTRE\(_{w/o\text{LTH}}\) | 85.0 ± 0.2    | 87.3 ± 0.3     | 87.6 ± 0.2      | 86.4 ± 0.3      | 87.5 ± 0.3 | 87.8 ± 0.3 | 86.2 ± 0.2 | 87.6 ± 0.2 | 88.0 ± 0.2 |
| CIFAR 10   | CTRE\(_{seq}\) | 47.9 ± 0.8      | 50.1 ± 0.5     | 50.1 ± 0.5      | 51.3 ± 0.5      | 52.7 ± 0.5 | 52.6 ± 0.6 | 52.0 ± 0.7 | 53.2 ± 0.6 | 53.6 ± 0.6 |
|            | CTRE\(_{w/o}\) | 48.2 ± 0.4      | 50.3 ± 0.3     | 50.5 ± 0.4      | 50.6 ± 0.4      | 52.6 ± 0.7 | 52.5 ± 0.7 | 50.0 ± 0.4 | 52.7 ± 0.5 | 53.5 ± 0.5 |
|            | CTRE\(_{w/o\text{Random}}\) | 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 |
|            | CTRE\(_{w/o\text{LTH}}\) | 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 |

Bold font indicates the best performer among the methods in the corresponding setting.
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 Fig. 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 Eq. 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 (Sect. 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: \[ \frac{A^{l}_{-p} \cdot A^{l}_{-q}}{C_{1}} \]. We evaluate the performance of this metric by replacing it with Eq. 2 in CTRE\(_{\text{sim}}\) and CTRE\(_{\text{seq}}\); we name these algorithms CTRE\(_{\text{sim-Hebb}}\) and 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 Sect. 4.2; we measure the performance on networks with different sizes and sparsity levels. The results are presented in Table 5 and the accuracy during training is plotted in Fig. 5. CTRE\(_{\text{sim-Hebb}}\) and CTRE\(_{\text{seq-Hebb}}\) have been outperformed by CTRE\(_{\text{sim}}\) and CTRE\(_{\text{seq}}\) 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.

The poor performance of CTRE\(_{\text{sim-Hebb}}\) and 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.

5.4 Convergence analysis

This section discusses the convergence of the proposed algorithm for training sparse neural networks from scratch, CTRE. In short, we first discuss the effect of the weight evolution process on the algorithm’s convergence. Secondly, we explore whether cosine similarity causes CTRE to converge into a local minimum or not.

First, we analyze if the weight evolution process in the CTRE algorithm interferes with the convergence of the back-propagation algorithm or not. In the CTRE algorithm, a number of connections of the weights are removed at each training epoch, and the same number of connections are added based on the cosine or random search policies. The weight evolution process is performed at each epoch after the standard feed-forward and back-propagation steps. The removed connections have a small magnitude compared to the other connections, and newly activated connections also get a small value. Therefore, they do not change the loss value significantly. The new weights will be updated in the next feed-forward and back-propagation step, and they will grow or shrink. Therefore, the weight evolution process does not disrupt the convergence of the model.
### Table 5 Classification accuracy (%) comparison of Cosine similarity-based methods and pure Hebbian-based evolution, on the Madelon dataset

| Dataset | Method      | $n' = 100$ | $n' = 500$ | $n' = 1000$ |
|---------|-------------|------------|------------|-------------|
|         | $\epsilon$ | $\epsilon$ | $\epsilon$ |             |
|         | $D(\%)$ ($N(\times 10^3)$) | $\epsilon$ | $\epsilon$ | $\epsilon$ |
| Madelon | CTRE$_{seq}$ | 82.2 ± 2.4 | 72.5 ± 2.0 | 63.9 ± 1.2 | 61.2 ± 2.4 | 81.5 ± 1.4 | 71.8 ± 2.0 | 61.1 ± 1.9 | 83.9 ± 2.0 | 76.5 ± 1.9 |
|         | CTRE$_{sim}$ | 81.6 ± 1.3 | 73.0 ± 1.6 | 65.6 ± 3.0 | 79.4 ± 1.7 | 77.7 ± 1.4 | 73.0 ± 1.5 | 78.8 ± 2.2 | 78.5 ± 1.0 | 74.6 ± 1.4 |
|         | CTRE$_{seq}$-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 |
|         | CTRE$_{sim}$-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 |

Bold font indicates the best performer among the methods in the corresponding setting.

Please note that $N$ (total number of parameters of the network) is scaled by $\times 10^3$. 


To validate this, we depict the test loss during training in Fig. 6 for the high sparsity regime and a large network ($\varepsilon = 1$, $n_l = 1000$). It can be observed that the loss function converges for the CTRE algorithm on all the datasets. In addition, in most cases, its convergence speed is much faster than for the other algorithms.

Secondly, we analyze whether CTRE is prone to converge to a local optima. As discussed in Sect. 5.2, cosine similarity is very successful at determining the most and least important connections in the network. However, in the mid-importance range, it might not be able to rank connections as well as the magnitude criterion; therefore, it might add some connections that do not contribute to decreasing the network loss. In such cases, the cosine similarity metric might prevent topology exploration and get stuck in local minima. CTRE explores other weights and exits this local minimum by using a random search. To validate this, in Fig. 7, we have presented the loss during training for CTRE seq, CTRE sim, and CTRE w/oRandom on three highly sparse neural networks trained on the Isolet dataset. The fast decrease in the loss in these plots indicates that all three methods quickly find a good-performing sub-network. However, the loss value of CTRE w/oRandom does not improve significantly after 200 epochs, and it converges to a higher value than the other two methods. Therefore, it is important to use random exploration to keep improving the topology and avoid local minima as it is done in CTRE.
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 (CTREseq) or simultaneously (CTRESim), 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

Fig. 6 Test loss comparison during training for the high sparsity regime and a large network \( \varepsilon = 1, n^l = 1000 \)

Fig. 7 Test loss comparison during training for the high sparsity regime \( \varepsilon = 1 \) on the Isolet dataset
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 (Pogodin et al., 2021). There have been some efforts to make CNNs more biologically plausible (Pogodin et al., 2021; Bartunov et al., 2018). Therefore, applying CTRE to CNNs should be done with great care and theoretical analysis that we believe is in the scope of future works.

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 Sect. 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 Sect. 4.2 in Figs. 8, 9, 10, 11, 12, and 13, which corresponds to Madelon, Isolet, MNIST, Fashion-MNIST, CIFAR10, and CIFAR100, 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:

\[
TD = \frac{\min_{acc \geq th \times acc_{max}, i \in \{1, 2, \ldots, \# \text{epochs}\}} \frac{i}{\# \text{epochs}}}{\# \text{epochs}},
\]

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 \(\varepsilon\), and \(\# \text{epochs}\) 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 6. 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 6, CTRE (including CTRE \(_{\text{sim}}\) and CTRE \(_{\text{seq}}\)) has the lowest training delay (\(TD\)) in 13 out of 18 cases considered. On the Isolet and the Madelon datasets, 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 a decent performance on Fashion-MNIST and MNIST, it has a poor performance on the other datasets. Finally, SET has comparable performance to other methods 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.

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 Eq. 2. Therefore, for each layer \(l\), CTRE requires in the order of \(O(mN^l)\) 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 $O(mN^l)$. 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 (CTRE$_{sample2}$) and a quarter of samples (CTRE$_{sample4}$) to compute the cosine similarity matrix. The results can be observed in Table 7. It is clear that even with half of the samples, CTRE can still achieve a close performance as the original method on all datasets. Based on these observations, it can be concluded that 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.

---

Fig. 9  Classification accuracy (%) results on Isolet
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. 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 Sect. 4.2 in the manuscript. The results are summarized in Table 8.

As can be seen in Table 8, the results are subtly different from Table 2 in Sect. 4.2. This difference arise from some small differences in the implementation. One of the main differences is that in Sect. 4.2, Tensorflow 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 policy. While in the experiments of Sect. 4.2, weights are initialized using a uniform distribution, in the sparse implemen-

Fig. 10 Classification accuracy (%) results on MNIST

---

4 The pure sparse implementation of SET can be found on https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks.
tation weights are initialized using a normal distribution which seems more beneficial to this implementation. Overall, in most cases, the results in Table 8 are higher than the results in Table 2.

**D Cosine vs. Euclidean-based similarity metric**

In this section, we analyze the effectiveness of cosine similarity metric in evolving the topology of the sparse neural network compared to other similarity metrics. To achieve this, we consider Euclidean-based similarity metric. Instead of computing the importance of the unexisting connections using cosine similarity (Eq. 2), we compute it as:

\[ \text{Sim}_{p,q} = \frac{1}{1 + d(A^{-1}_{p,q} \cdot A^{-1}_{q,p})} \]

where \( d(a, b) \) is the Euclidean distance between vectors \( a \) and \( b \). We replace Eq. 4 with Eq. 2 in Algorithm 2 and we call this method as CTRE\textsubscript{sim-euclidean}. We compare CTRE\textsubscript{sim-euclidean} with CTRE\textsubscript{sim}. The results are presented in Table 9.

As shown in Table 9, CTRE\textsubscript{sim} outperforms CTRE\textsubscript{sim-euclidean} in most cases considered. Particularly, in the high sparsity regime (\( \varepsilon = 1 \)), there is a considerable gap between the
It can be concluded that Euclidean-based similarity metric is not very informative in evolving the topology of sparse neural networks. This might stem from the sensitivity of this metric to the vectors’ magnitude. Cosine-similarity metric, a magnitude insensitive metric which also presents a biologically-plausible approach for measuring the importance of the connections, is a good choice for obtaining sparse neural networks in the CTRE algorithm.

**E Ablation study: cosine and random-based weight addition order in CTRE\textsubscript{seq}**

In this section, we perform an ablation study on the CTRE\textsubscript{seq} algorithm to measure the effectiveness of the cosine-based and random search order in the performance of the algorithm. With this aim, we measure the performance of two variants of CTRE\textsubscript{seq} (Algorithm 1) that are different from CTRE\textsubscript{seq} in the order of the cosine and random weight addition:

- **CTRE\textsubscript{ablv1.seq}** starts training with random weight addition, and switches to cosine-based addition when there is no improvement in the validation accuracy for \(\epsilon_{early\ stop}\) epochs.

---

![Classification accuracy (%) results on CIFAR10](image-url)
CTRE abl 2:seq splits the training process into two equal phases in terms of the number of epochs. In the first phase it adds weights randomly, and in the second phase it uses the cosine-similarity information for weight addition.

We compare the performance of these two algorithms with CTRE seq in Table 10. CTRE seq outperforms CTRE abl1:seq and CTRE abl2:seq in the majority of the experiments. In high sparsity regime and large network size ($\epsilon = 1, n_l = 1000$), there is a considerable gap between their performance. The reason behind this difference is that by using cosine-similarity-based weight addition at the beginning of the algorithm, the algorithm finds a well-performing sub-network very fast. Then, in the rest of the algorithm it improves this...
Table 6  Training delay (TD) (%) comparison among methods

| Method      | Madelon | Isolet | MNIST | Fashion-MNIST | CIFAR10 | CIFAR100 |
|-------------|---------|--------|-------|---------------|---------|----------|
|              | n'/n'   | n'/n'  | n'/n' | n'/n'         | n'/n'   | n'/n'    |
| ε            | 100     | 500    | 1000  | 100           | 500     | 1000     |
| SNIP        | 1.8     | 2.8    | 3.2   | 5.6           | 8.8     | 8.6      |
| RigL        | 3.2     | 5.8    | 9.6   | 2.8           | 3.6     | 4.6      |
| 1 SET       | 40.0    | 74.0   |       | 4.4           | 11.8    | 18.2     | 11.4     | 39.4    | 68.4    | 40.2    | 16.8    | 40.4    | 36.6    |
| CTRExeq     | 16.0    |        | 48.0  | 52.8          | 52.4    | 2.2      | 2.6     | 3.0      | 3.6     | 9.2     | 12.8     | 13.4     | 14.6    | 16.8    | –       | 40.4    | 36.6    |
| CTRExsim    | 9.8     | 20.6   | 24.8  | 38.6          | 30.6    | 35.6     | 2.2     | 3.0      | 3.2     | 3.0     | 6.2     | 7.2      | 11.4     | 22.4    | 33.2    | 66.8    | 88.6    | –       |

Empty fields indicate that the method cannot reach the considered level of accuracy in 500 training epochs.
| Dataset        | Method       | $n' = 100$ | $n' = 500$ | $n' = 1000$ |
|---------------|--------------|------------|------------|-------------|
|               | $\epsilon$  | 1          | 5          | 13          | 1          | 5          | 13          | 1          | 5          | 13          |
| Madelon       | CTRE$_{sam}$ | 81.6 ± 1.3 | 73.0 ± 1.6 | 65.6 ± 3.0 | 79.4 ± 1.7 | 77.7 ± 1.4 | 73.0 ± 1.5 | 78.8 ± 2.2 | 78.5 ± 1.0 | 74.6 ± 1.4 |
|               | CTRE$_{sam}k2$ | 81.0 ± 2.3 | 72.4 ± 1.4 | 65.3 ± 1.6 | 79.7 ± 0.5 | 77.9 ± 0.9 | 71.7 ± 2.8 | 80.6 ± 0.7 | 78.6 ± 1.0 | 74.1 ± 0.4 |
|               | CTRE$_{sam}k4$ | 81.9 ± 1.0 | 71.3 ± 1.8 | 64.4 ± 1.0 | 79.0 ± 1.1 | 77.8 ± 0.8 | 72.7 ± 1.6 | 78.7 ± 1.1 | 79.1 ± 1.4 | 74.1 ± 0.7 |
| Isolet        | CTRE$_{sam}$ | 87.5 ± 0.8 | 93.4 ± 0.7 | 94.3 ± 0.8 | 87.8 ± 1.1 | 91.7 ± 1.1 | 93.9 ± 0.5 | 88.3 ± 0.7 | 91.3 ± 1.3 | 93.1 ± 0.6 |
|               | CTRE$_{sam}k2$ | 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 |
|               | CTRE$_{sam}k4$ | 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         | CTRE$_{sam}$ | 95.5 ± 0.2 | 97.3 ± 0.1 | 97.8 ± 0.1 | 96.4 ± 0.2 | 97.7 ± 0.1 | 98.0 ± 0.1 | 96.6 ± 0.2 | 97.7 ± 0.1 | 97.9 ± 0.1 |
|               | CTRE$_{sam}k2$ | 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 |
|               | CTRE$_{sam}k4$ | 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 | CTRE$_{sam}$ | 85.8 ± 0.3 | 87.5 ± 0.3 | 88.0 ± 0.2 | 86.4 ± 0.5 | 88.1 ± 0.2 | 88.3 ± 0.2 | 86.5 ± 0.3 | 88.1 ± 0.3 | 88.3 ± 0.2 |
|               | CTRE$_{sam}k2$ | 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 |
|               | CTRE$_{sam}k4$ | 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       | CTRE$_{sam}$ | 48.2 ± 0.4 | 50.3 ± 0.3 | 50.5 ± 0.4 | 50.6 ± 0.4 | 52.6 ± 0.7 | 52.5 ± 0.7 | 50.0 ± 0.4 | 52.7 ± 0.5 | 53.5 ± 0.5 |
|               | CTRE$_{sam}k2$ | 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 |
|               | CTRE$_{sam}k4$ | 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 |
### Table 8 Classification accuracy (%) comparison using pure sparse implementation

| Dataset    | Method       | $n' = 100$ |       | $n' = 500$ |       | $n' = 1000$ |       |
|------------|--------------|------------|-------|------------|-------|------------|-------|
|            | $\varepsilon$ | 1          | 5     | 13         | 1     | 5          | 13    |
| Madelon    | 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 |
|            | CTREnwRandom | 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 |
|            | CTREn_seq    | 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 |
|            | CTREn_im     | **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     | 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 |
|            | CTREnwRandom | 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** |
|            | CTREn_seq    | 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 |
|            | CTREn_im     | 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      | 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 |
|            | CTREnwRandom | **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** |
|            | CTREn_seq    | 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 |
|            | CTREn_im     | 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 | 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 |
|            | CTREnwRandom | 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** |
|            | CTREn_seq    | 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 |
|            | CTREn_im     | 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 |
| CIFAR 10   | 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 |
|            | CTREnwRandom | **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** |
|            | CTREn_seq    | **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 |
|            | CTREn_im     | 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 |
| Dataset          | Method                  | $n' = 100$                     | $n' = 500$                     | $n' = 1000$                   |
|------------------|-------------------------|--------------------------------|--------------------------------|--------------------------------|
|                  | $\epsilon$              | 1  | 5  | 13 | 1  | 5  | 13 | 1  | 5  | 13 |
| Madelon          | CTRE_{sim}              | 81.6 ± 1.3  | 73.0 ± 1.6  | 65.6 ± 3.0  | 79.4 ± 1.7  | 77.7 ± 1.4  | 73.0 ± 1.5  | 78.8 ± 2.2  | 78.5 ± 1.0  | 74.6 ± 1.4 |
|                  | CTRE_{sim-euclidean}    | 77.2 ± 1.7  | 65.6 ± 1.3  | 63.7 ± 0.7  | 71.5 ± 4.2  | 74.2 ± 1.6  | 65.8 ± 4.3  | 70.6 ± 4.5  | 76.9 ± 1.8  | 67.8 ± 2.2 |
| Isolet           | CTRE_{sim}              | 87.5 ± 0.8  | 93.4 ± 0.7  | 94.3 ± 0.8  | 87.8 ± 1.1  | 91.7 ± 1.1  | 93.9 ± 0.5  | 88.3 ± 0.7  | 91.3 ± 1.3  | 93.1 ± 0.6 |
|                  | CTRE_{sim-euclidean}    | 83.0 ± 1.3  | 93.7 ± 0.6  | 94.5 ± 0.7  | 72.0 ± 4.1  | 91.6 ± 0.3  | 93.7 ± 0.8  | 49.8 ± 6.1  | 91.4 ± 0.7  | 93.6 ± 0.5 |
| MNIST            | CTRE_{sim}              | 95.5 ± 0.2  | 97.3 ± 0.1  | 97.8 ± 0.1  | 96.4 ± 0.2  | 97.7 ± 0.1  | 98.0 ± 0.1  | 96.6 ± 0.2  | 97.7 ± 0.1  | 97.9 ± 0.1 |
|                  | CTRE_{sim-euclidean}    | 94.6 ± 0.2  | 97.0 ± 0.1  | 97.3 ± 0.1  | 94.3 ± 0.2  | 97.2 ± 0.1  | 97.4 ± 0.0  | 93.8 ± 0.1  | 96.9 ± 0.0  | 97.1 ± 0.3 |
| Fashion-MNIST    | CTRE_{sim}              | 85.8 ± 0.3  | 87.5 ± 0.3  | 88.0 ± 0.2  | 86.4 ± 0.5  | 88.1 ± 0.2  | 88.3 ± 0.2  | 86.5 ± 0.3  | 88.1 ± 0.3  | 88.3 ± 0.2 |
|                  | CTRE_{sim-euclidean}    | 84.9 ± 0.2  | 87.1 ± 0.1  | 87.7 ± 0.1  | 84.6 ± 0.2  | 87.4 ± 0.2  | 87.8 ± 0.2  | 83.8 ± 0.3  | 87.4 ± 0.1  | 87.4 ± 0.4 |
| CIFAR10          | CTRE_{sim}              | 48.2 ± 0.4  | 50.3 ± 0.3  | 50.5 ± 0.4  | 50.6 ± 0.4  | 52.6 ± 0.7  | 52.5 ± 0.7  | 50.0 ± 0.4  | 52.7 ± 0.5  | 53.5 ± 0.5 |
|                  | CTRE_{sim-euclidean}    | 46.1 ± 0.4  | 50.1 ± 0.4  | 50.7 ± 0.4  | 46.1 ± 1.4  | 50.5 ± 0.5  | 51.6 ± 0.2  | 44.8 ± 0.5  | 49.5 ± 0.6  | 51.2 ± 0.3 |
| CIFAR100         | CTRE_{sim}              | 13.8 ± 0.4  | 20.6 ± 0.4  | 21.9 ± 0.4  | 17.0 ± 0.3  | 23.0 ± 0.3  | 24.6 ± 0.4  | 17.3 ± 0.4  | 23.5 ± 0.3  | 25.1 ± 0.3 |
|                  | CTRE_{sim-euclidean}    | 12.2 ± 0.4  | 20.7 ± 0.4  | 21.8 ± 0.3  | 11.8 ± 0.7  | 22.5 ± 0.3  | 24.0 ± 0.2  | 9.4 ± 1.0   | 21.9 ± 0.3  | 23.8 ± 0.4 |
| Dataset    | Method       | $n' = 100$      | $n' = 500$      | $n' = 1000$     |
|------------|--------------|-----------------|-----------------|----------------|
|            |              | $\epsilon$      | $\epsilon$      | $\epsilon$     |
| Madelon    | CTRE$_{seq}$ | 82.2 ± 2.4      | 72.5 ± 2.0      | 63.9 ± 1.2     | 61.2 ± 2.4      | 81.5 ± 1.4      | 71.8 ± 2.0     | 61.1 ± 1.9      | 83.9 ± 2.0      | 76.5 ± 1.9     |
|            | CTRE$_{ab11:seq}$ | 57.8 ± 2.3      | 58.1 ± 3.1      | 57.8 ± 2.1     | 61.0 ± 3.1      | 59.6 ± 3.0      | 59.4 ± 2.0     | 61.1 ± 2.9      | 59.7 ± 1.1      | 58.3 ± 2.4     |
|            | CTRE$_{ab12:seq}$ | 59.2 ± 3.0      | 58.0 ± 3.5      | 58.4 ± 2.3     | 61.1 ± 2.0      | 58.7 ± 2.9      | 58.8 ± 2.4     | 61.6 ± 0.9      | 58.4 ± 1.6      | 58.7 ± 1.9     |
| Isolet     | CTRE$_{seq}$ | 86.7 ± 1.8      | 92.4 ± 1.1      | 94.3 ± 0.5     | 87.2 ± 2.0      | 92.3 ± 0.6      | 94.0 ± 0.4     | 86.7 ± 1.6      | 91.5 ± 1.0      | 93.7 ± 0.5     |
|            | CTRE$_{ab11:seq}$ | 89.5 ± 1.1      | 93.7 ± 0.5      | 94.9 ± 0.5     | 85.8 ± 1.0      | 93.3 ± 0.8      | 94.4 ± 0.4     | 72.5 ± 2.3      | 92.3 ± 0.6      | 93.5 ± 0.9     |
|            | CTRE$_{ab12:seq}$ | 89.0 ± 1.3      | 93.8 ± 0.6      | 94.6 ± 0.7     | 85.8 ± 1.5      | 93.1 ± 0.8      | 94.2 ± 0.9     | 75.1 ± 3.9      | 92.4 ± 0.9      | 93.3 ± 0.7     |
| MNIST      | CTRE$_{seq}$ | 95.7 ± 0.2      | 97.3 ± 0.2      | 97.7 ± 0.1     | 97.0 ± 0.2      | 97.6 ± 0.2      | 97.8 ± 0.1     | 97.3 ± 0.1      | 97.7 ± 0.1      | 97.8 ± 0.1     |
|            | CTRE$_{ab11:seq}$ | 95.5 ± 0.3      | 97.1 ± 0.1      | 97.6 ± 0.1     | 95.8 ± 0.2      | 97.3 ± 0.1      | 97.5 ± 0.1     | 95.9 ± 0.1      | 97.3 ± 0.1      | 97.2 ± 0.1     |
|            | CTRE$_{ab12:seq}$ | 95.5 ± 0.2      | 97.1 ± 0.1      | 97.6 ± 0.1     | 96.1 ± 0.1      | 97.4 ± 0.2      | 97.5 ± 0.1     | 96.0 ± 0.1      | 97.4 ± 0.1      | 97.3 ± 0.0     |
| Fashion-MNIST | CTRE$_{seq}$ | 85.8 ± 0.5      | 87.5 ± 0.3      | 87.4 ± 0.3     | 87.1 ± 0.4      | 87.9 ± 0.3      | 88.0 ± 0.2     | 87.3 ± 0.2      | 88.0 ± 0.3      | 88.3 ± 0.2     |
|            | CTRE$_{ab11:seq}$ | 85.7 ± 0.2      | 87.5 ± 0.2      | 87.8 ± 0.2     | 86.3 ± 0.3      | 87.7 ± 0.2      | 87.9 ± 0.3     | 86.0 ± 0.2      | 87.7 ± 0.3      | 87.5 ± 0.3     |
|            | CTRE$_{ab12:seq}$ | 85.9 ± 0.3      | 87.4 ± 0.2      | 87.9 ± 0.2     | 86.4 ± 0.2      | 87.7 ± 0.0      | 87.2 ± 0.2     | 86.2 ± 0.1      | 87.7 ± 0.1      | 87.6 ± 0.2     |
| CIFAR 10   | CTRE$_{seq}$ | 47.9 ± 0.8      | 50.1 ± 0.5      | 50.1 ± 0.5     | 51.3 ± 0.5      | 52.7 ± 0.5      | 52.6 ± 0.6     | 52.0 ± 0.7      | 53.2 ± 0.6      | 53.6 ± 0.6     |
|            | CTRE$_{ab11:seq}$ | 48.1 ± 0.4      | 49.3 ± 0.6      | 50.5 ± 0.4     | 49.3 ± 0.4      | 50.9 ± 0.4      | 51.1 ± 0.5     | 48.9 ± 0.4      | 50.9 ± 0.4      | 50.8 ± 0.7     |
|            | CTRE$_{ab12:seq}$ | 47.8 ± 0.2      | 49.9 ± 0.4      | 50.3 ± 0.2     | 49.2 ± 0.4      | 50.8 ± 0.4      | 51.1 ± 0.2     | 48.6 ± 0.1      | 51.0 ± 0.1      | 50.7 ± 0.3     |
| CIFAR 100  | CTRE$_{seq}$ | 12.7 ± 0.7      | 21.1 ± 0.3      | 21.8 ± 0.5     | 18.7 ± 0.4      | 23.6 ± 0.4      | 24.0 ± 0.4     | 21.4 ± 0.4      | 23.9 ± 0.5      | 24.7 ± 0.4     |
|            | CTRE$_{ab11:seq}$ | 14.9 ± 0.6      | 20.5 ± 0.3      | 21.5 ± 0.5     | 13.9 ± 2.3      | 22.8 ± 0.3      | 23.8 ± 0.6     | 1.0 ± 0.0       | 23.6 ± 0.3      | 24.3 ± 0.4     |
|            | CTRE$_{ab12:seq}$ | 14.5 ± 0.3      | 20.4 ± 0.1      | 21.4 ± 0.3     | 14.1 ± 0.9      | 22.8 ± 0.0      | 24.0 ± 0.1     | 1.0 ± 0.0       | 23.4 ± 0.4      | 24.6 ± 0.2     |
topology using cosine information and random search. However, by starting with random weight addition, it might take longer for the algorithm to reach a reasonable level of performance. This results in a lower accuracy than CTRE_seq at the end of training.

To summarize, while starting with cosine similarity-based weight addition and switching to random search might seem counter-intuitive, we show that this strategy is beneficial for the CTRE algorithm.

Author Contributions JP and DCM designed and developed a preliminary idea and proof of concept experiments. ZA designed and developed the CTRE algorithm, designed and carried out the empirical validation, and performed the analysis and interpretation of the results. SL helped in performing the experiments and interpreting the results. DCM, RV and MP supervised the research. All authors provided critical feedback and helped shaped the research. ZA wrote the manuscript with input from all authors.

Funding This project is partially funded by the NWO EDIC project.

Availability of data and material All the datasets used in this research are currently public. The references for all datasets have been cited in Sect. 4.1.4.

Declarations

Conflicts of interest Not applicable.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Code availability The implementation code is available on Github at https://github.com/zahraatashgahi/CTRE.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. https://www.tensorflow.org/. Software available from tensorflow.org.

Arora, S., Bhaskara, A., Ge, R., & Ma, T. (2014). Provable bounds for learning some deep representations. In International conference on machine learning (pp. 584–592). PMLR, 2014.

Atashgahi, Z., Sokar, G., van der Lee, T., Mocanu, E., Mocanu, D. C., Veldhuis, R., & Pechenizkiy, M. (2022). Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. Machine Learning (ECML-PKDD 2022 journal track) 1–38.

Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., & Lillicrap, T. (2018). Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In Proceedings of the 32nd international conference on neural information processing systems (pp. 9390–9400).
Bellec, G., Kappel, D., Maass, W., & Legenstein, R. (2018). Deep rewiring: Training very sparse deep networks. In International conference on learning representations. https://openreview.net/forum?id=BJ4wN01C-

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D, Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. & Lin,H. (eds.), Advances in neural information processing systems (Vol. 33, pp. 1877–1901). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc649674188f88ac142f64a-Paper.pdf.

Dai, X., Yin, H., & Jha, N.K. (2019). Nest: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers, 68(10):1487–1497.

de Jorge, P., Sanyal, A., Behl, H.S, Torr, P.H.S., Rogez, G., & Dokania, P.K. (2020). Progressive skeletonization: Trimming more fat from a network at initialization. arXiv preprint arXiv:2006.09081.

Dettmers, T., & Zettlemoyer, L. (2019). Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840.

Evci, U., Gale, T., Menick, J., Castro, P. S., & Elsen, E. (2020). Rigging the lottery: Making all tickets winners. In International conference on machine learning (pp. 2943–2952). PMLR, 2020.

Fanty, M., & Cole, R. (1991). Spoken letter recognition. In Advances in neural information processing systems (pp. 220–226).

Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.

Friston, K. (2008). Hierarchical models in the brain. PLoS Computational Biology. 4(11), e1000211.

Gale, T., Elsen, E., & Hooker, S.(2019). The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574.

Galle, L., & Scherp, A. (2021). Forget me not: A gentle reminder to mind the simple multi-layer perceptron baseline for text classification. arXiv preprint arXiv:2109.03777.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., & Choi, E. (2018). Morphnet: Fast & simple resource-constrained structure learning of deep networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1586–1595).

Gorishniy, Y., Rubacev, I., Khrulkov, V., & Babenko, A. (2021). Revisiting deep learning models for tabular data. arXiv preprint arXiv:2106.11959.

Graves, A., Mohamed, A.-R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645–6649). IEEE.

AI High-Level Expert Group. (2020). Assessment list for trustworthy artificial intelligence (ALTAI) for self-assessment.

Guo, Y., Yao, A., & Chen, Y.(2016). Dynamic network surgery for efficient dnn training in Proceedings of the 30th international conference on neural information processing systems, NIPS’16 (pp. 1387-1395). Red Hook, NY: Curran Associates Inc. ISBN 9781510838819.

Guyon, I., Gunn, S., Nikravesh, M., & Zadeh, L.A. (2008). Feature extraction: Foundations and applications (Vol. 207). Springer.

Han, J., Kamber, M., & Pei, J., et al. (2012). Getting to know your data. Data mining (pp. 39–82). Netherlands: Elsevier Amsterdam.

Han, S., Pool, J., Tran, J., & Dally, W.J. (2015). Learning both weights and connections for efficient neural networks. In Proceedings of the 28th international conference on neural information processing systems (Vol. 1, pp. 1135–1143).

Hassibi, B., & Stork, D.G.(1993). Second order derivatives for network pruning: Optimal brain surgeon. In Advances in neural information processing systems (pp. 164–171).

Hebb, D.O. (2005). The organization of behavior: A neuropsychological theory. Psychology Press.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kiamehrad, H., Patwary, M., Ali, M., Yang, Y., & Zhou, Y. (2017). Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. arXiv preprint arXiv:2102.00554.

Jayakumar, S., Pascaru, R., Rae, J., Osindero, S., & Elsen, E. (2020). Top-kast: Top-k always sparse training. Advances in Neural Information Processing Systems. 33. 20744–20754.

Jouppi, N.P, Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., & Borchers, A. et al.(2017). In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th annual international symposium on computer architecture (pp. 1–12).
Liang, M., Liu, C., Li, Y., Gu, S., Mayer, C., Gool, L.V., Mocanu, D.C., Mocanu, E., Nguyen, P.H., Gibescu, M., & Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Madeleine, G., Masi, I., Wu, Y., Hassner, T., Li, B., LeCun, Y., Denker, J.S., & Lee, N., Ajanthan, T., http://yann.lecun.com/exdb/mnist/.

LeCun, Y. (1998). The mnist database of handwritten digits. In Proceedings of the IEEE conference on computer vision and pattern recognition (Vol. 119, pp. 5544–5555). http://proceedings.mlr.press/v119/lecun19a.html.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In Machine learning proceedings 1995 (pp. 331–339). Elsevier.

LeCun, Y. (1998). The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

LeCun, Y., Denker, J.S., & Solla, S.A. (1990). Optimal brain damage. In Advances in neural information processing systems (pp. 598–605).

Lee, N., Ajanthan, T., & Torr, P.(2019). SNIP: Single-shot network pruning based on connection sensitivity. In International conference on learning representations. https://openreview.net/forum?id=B1VZqjAcYX.

Li, B., & Han, L. (2013). Distance weighted cosine similarity measure for text classification. In International conference on intelligent data engineering and automated learning (pp. 611–618). Springer.

Li, Y., Gu, S., Mayer, C., Gool, L.V., & Timofte, R. (2020). 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 (pp. 8018–8027).

Liang, M., & Hu, X.(2015). Recurrent convolutional neural network for object recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3367–3375).

Liu, C., & Wu, H. (2019). Channel pruning based on mean gradient for accelerating convolutional neural networks. Signal Processing, 156(84–91), 2019.

Liu, J., Gong, M., & Miao, Q. (2017). Modeling hebb learning rule for unsupervised learning. In IJCAI (pp. 2315–2321).

Liu, S., van der Lee, T., Yaman, A., Atashgahi, Z., Ferrar, D., & Sokar, G., et al. (2020). 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) (pp. 2006–14085).

Liu, S., Mocanu, D.C., Matavalam, A. R. R., Pei, Y., & Pechenizkiy, M. (2021). Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. Neural Computing and Applications, 33(7), 2589–2604.

Liu, S., Mocanu, D.C., Pei, Y., & Pechenizkiy, M. (2021b). Selfish sparse rnn training. In Marina, M., & Tong, Z. (eds), Proceedings of the 38th international conference on machine learning (Vol. 139, pp. 6893–6904). https://proceedings.mlr.press/v139/liu21p.html.

Liu, S., Yin, L., Mocanu, D.C., & Pechenizkiy, M. (2021c). Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In Marina, M., & Tong, Z. (eds), Proceedings of the 38th international conference on machine learning (Vol.139, pp. 6989–7000). https://proceedings.mlr.press/v139/liu21y.html.

Louizos, C., Welling, C., & Kingma, D.P. (2018). Learning sparse neural networks through l0 regularization. In International conference on learning representations. https://openreview.net/forum?id=H1Y8hhg0b.

Luo, C., Zhan, J., Xue, X., Wang, L., Ren, R., & Yang, Q. (2018). Cosine normalization: Using cosine similarity instead of dot product in neural networks. In International conference on artificial neural networks (pp. 382–391). Springer.

Masi, I., Wu, Y., Hassner, T., & Natarajan, P. (2018). Deep face recognition: A survey. In 2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI) (pp. 471–478). IEEE.

Mocanu, D.C., Mocanu, E., Nguyen, P.H., Gibescu, M., & Liotta, A. (2016). A topological insight into restricted boltzmann machines. Machine Learning, 104(2–3), 243–270.

Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Madeleine, G., & Antonio, L. (2018). Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nature Communications, 9(1), 2383.

Mocanu, D.C., Mocanu, E., Pinto, T., Curci, S., Nguyen, P.H., Gibescu, M., Ernst, D., & Vale, Z.A. (2021). Sparse training theory for scalable and efficient agents. In Proceedings of the 20th international conference on autonomous agents and multiagent systems (pp. 34–38).
Molchanov, Dmitry, A., & Arsenii, V. D. (2017). Variational dropout sparsifies deep neural networks. In *International conference on machine learning* (pp. 2498–2507). PMLR.

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., & Kautz, J. (2019). Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, June.

Mostafa, H., & Wang, X. (2019). Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In Kamalika, C., & Ruslan, S. (eds), *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 4646–4655). http://proceedings.mlr.press/v97/mostafa19a.html.

NEYSHABUR, B., Li, Z., Bhojanapalli, S., LeCun, Y., & Srebro, N. (2019). The role of over-parametrization in generalization of neural networks. In *International conference on learning representations*. https://openreview.net/forum?id=BygIghAcYX.

Nguyen, H.V., & Bai, L. (2010). Cosine similarity metric learning for face verification. In *Asian conference on computer vision* (pp. 709–720). Springer.

Pogodin, R., Mehta, Y., Lillicrap, T.P., & Latham, P.E. (2021). Towards biologically plausible convolutional networks. arXiv preprint arXiv:2106.13031.

Popov, S., Morozov, S., & Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data. arXiv preprint arXiv:1909.06312.

Raithan, M.A., & Aamodt, T.M. (2020) Sparse weight activation training. arXiv preprint arXiv:2001.01969.

Savarese, P., Silva, H., & Maire, M. (2020). Winning the lottery with continuous sparsification. In H. Larochelle, M. Ranzato, P. Hadsell, M. F. Balcan, & H. Lin (eds.), *Advances in neural information processing systems* (Vol. 33, pp. 11380–11390). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/83004190b1793d7aa15f8d0d49a13eba-Paper.pdf.

Scellier, B., & Bengio, Y. (2016). Towards a biologically plausible backprop. arXiv preprint arXiv:1602.05179.

Schuchamher, T.(2021). Live wired neural networks: Making neurons that fire together wire together. arXiv preprint arXiv:2105.08111.

Sidorov, G., Gelbukh, A., Gómez-Adorno, H., & Pinto, D. (2014). Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas, 18*(3), 491–504.

Sun, Y., Wang, X., & Tang, X.(2016). Sparsifying neural network connections for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4856–4864).

Tanaka, H., Kunin, D., Yamins, D. L., & Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems, 33*, 6377–6389.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhang, X., Unterthiner, T., Yang, J., Keysers, D., Uszkoreit, J., & Lueic, M., et al.(2021). Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601.

Wang, C., Grosse, R., Fidler, S., & Zhang, G. (2019a). Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International conference on machine learning* (pp. 6566–6575). PMLR.

Wang, C., Zhang, G., & Grosse, R. (2019). Picking winning tickets before training by preserving gradient flow.

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. In *Proceedings of the 30th international conference on neural information processing systems, NIPS’16* (pp. 2082-2090). Red Hook, NY: Curran Associates Inc.

Xia, P., Zhang, L., & Li, F. (2015). Learning similarity with cosine similarity ensemble. *Information Sciences*, 307:39–52. ISSN 0020-0255. https://doi.org/10.1016/j.ins.2015.02.024. URL https://www.sciencedirect.com/science/article/pii/S0020025515001243.

Xiao, H., Rasul, K., & Vollgraf, R.(2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms.

Yang, J., Xiao, W., Jiang, C., Hosain, M. S., Muhammad, G., & Amin, S. U. (2018). Ai-powered green cloud and data center. *IEEE Access, 7*, 4195–4203.

Zhang, M., Zhang, F., Lane, N. D., Shu, Y., Zeng, X., & Fang, B., et al. (2020). *Deep learning in the era of edge computing: Challenges and opportunities* (p. 20). Fog Computing: Theory and Practice.

Zhu, M., & Gupta, S.(2017). To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878.

**Publisher’s Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.