ORIGINAL ARTICLE

Multilayer perceptron ensembles in a truly sparse training context

Peter R. D. van der Wal[1] · Nicola Strisciuglio[2] · George Azzopardi[1] · Decebal Constatin Mocanu[3]

Received: 25 October 2024 / Accepted: 16 April 2025 / Published online: 23 May 2025

Abstract
Ensemble learning for artificial neural networks (ANNs) is an effective method to enhance predictive performance. However, ANNs are computationally and memory intensive, and naively training multiple networks can lead to excessive training times and costs. An effective tool for improving ensemble efficiency is introducing topological sparsity. Even though several implementations of efficient ensembles have been proposed, none of them can provide actual benefits in terms of computational overhead as the sparsity is simulated using binary masks. In this paper, we address this issue by introducing a Truly Sparse Ensemble without binary masks and directly incorporate native sparsity. We also propose two algorithms for initializing new subnetworks within the ensemble, leveraging this native topological sparsity to enhance subnetwork diversity. We demonstrate the performance of the resulting models at high levels of sparsity on several datasets in terms of classification accuracy, floating point operations (FLOPs), and actual running time. The proposed methods outperform all baseline dense and truly sparse models on tabular data, successfully diversify the training trajectory of the subnetworks, and increase the topological distance between subnetworks after re-initialization.

Keywords Truly sparse training · MLP · Ensemble training · Diversity

1 Introduction

Artificial neural networks (ANNs) have driven intense recent advancements in machine learning and related fields research. Due to the ever-increasing amount of available data and the growing complexity of problems, the demanded capability of neural networks is growing as well. Larger networks are trained, which risk to overfit training data due to overparametrization. An architectural method to improve performance without risking overfitting and significantly increasing computational overhead is ensemble learning [1]. The core idea of this paradigm is to ensemble several weak learners which are computationally cheap to obtain a strong combined learner.

For ANNs with fully connected neurons, the number of connections between consecutive layers increases quadratically with the number of neurons in the respective layers. The resources needed for training and applying deep neural networks are thus often prohibitive [2]. In contrast, neurophysiological findings about the topology of the brain, as showed by [3], reveals that the more neurons a human brain has, the fewer connections between neurons are created. By transferring this concept to ANNs, [4] showed that sparse networks could obtain the same level of accuracy as their dense counterparts.

https://doi.org/10.1007/s00521-025-11294-3

In the realm where sparsity intersects with ensemble learning, the concept of sparse ensembles emerges. Although there has been research on pre-defined sparsity and pruning weights during training, little to no practical advantages are obtained as most sparse training is simulated by applying binary masks to the weight matrices. Despite allowing to study the behavior of sparse ensembles, the masked-sparsity approach does not decrease the computation and memory overhead. The reason for existing works to mostly adopt masked sparsity is that almost all specialized deep learning frameworks and hardware are optimized for dense matrix operations [5]. Thus, developing sparse deep neural networks using sparse matrix operations is much more complicated than sparse networks that leverage standard (dense matrix) deep learning libraries. [6] introduced a truly sparse multilayer perceptron (MLP) and trained a neural network with more than one million neurons on a regular CPU without additional GPU support. Together with hardware improvements, algorithmic and software developments for truly sparse training are vital to actually provide faster, energy-efficient, and memory-efficient deep neural networks.

In this paper, we demonstrate the training of high-efficiency sparse ensembles in the *unstructured sparsity setting*, without the overhead of binary masks, and propose methods to control subnetwork diversification in sparse ensembles to improve overall performance.

We introduce a first-of-its-kind Truly Sparse Ensemble and train it on just one single CPU core at very high levels of sparsity. The main focus of this work is on tabular data and the MLP architecture that we train in our ensemble, as it is commonly used in low-resource applications such as IoT [7] and computation at the edge [8]. The main contributions include:

- The development of a Truly Sparse MLP Ensemble, further demonstrating the advantages of the truly sparse framework during training in terms of performance, number of floating point operations, and running time.
- Two efficient and effective subnetwork diversification strategies regrow subnetwork weights based on the Euclidean distance to its predecessor or the topological dissimilarity to all predecessors, respectively.

2 Related work

2.1 Sparse training

Over the past few years, sparse implementations for several ANN architectures have been introduced including Restricted Boltzmann Machines [9], Convolutional Neural Networks [10–13], and Recurrent Neural Networks [12]. Enforced sparsity levels vary greatly and can range from a moderate sparsity level of 50% as in [14], to extreme sparsity levels (>99%) as presented in [15]. A distinction between two main approaches to realize unstructured sparsity in an ANN can be made, namely dense-to-sparse training and sparse-to-sparse training. Dense-to-sparse training concerns pruning of the non-critical connections of the ANN during training with a dense network as a starting point (dense-to-sparse training) [4, 16–18]. In sparse-to-sparse training, a sparsely initialized network is either trained with a fixed sparse topology [9, 19] or a dynamically updating topology by pruning and regrowing weights during training [20, 21] (sparse-to-sparse training). Dynamic Sparse Training (DST) has gained a lot of popularity and has led to the introduction of many new DST-based methods, such as the Sparse Evolutionary Training (SET) [20] and Rigged Lottery (RigL) [22], using various model types, pruning criteria, and regrowth methods [20, 22–26]. A benefit of these methods is the reduced memory overhead as a result of sparse initialization. Nevertheless, none of these methods provide actual benefit in terms of efficiency as the sparsity is simulated using binary masks, usually causing additional computational and memory overhead.

2.2 Efficient ensembles

The fundamental concept of ensemble methods lies in integrating multiple weak learners, which together can achieve superior outcomes compared to a single, standalone learner. Ensemble members can be obtained in

parallel (independently) or sequentially [27, 28]. Diversity among such ensemble members is key for the effectiveness of the overall ensemble for several reasons, primarily because it facilitates the exploration of a larger part of the solution space [1, 28–34]. Methods to establish diversity in ensembles vary depending on whether the ensemble is obtained in parallel or sequentially. In parallel ensembles, diversity is often introduced through data diversity techniques such as (feature space) bagging [1, 35–38], which trains independent models on different subsets of data. In contrast, sequential ensembles typically leverage boosting [39, 40], where later models focus on correcting errors made by earlier ones, enhancing diversity through iterative learning. Structural diversity methods, which incorporate different models as base predictors [1, 29, 41–43], and parameter diversity methods, such as evolutionary and cross-over learning [32, 44], can be applied both sequentially and in parallel and further contribute to the diversity of the ensemble. The recent work of [45] demonstrated that, unlike ensembles with small subnetworks, ensembles consisting of very large subnetworks, such as ResNet-18, do not benefit from actively promoting predictive diversity. Topological sparsity can be used to create structural diversity among subnetworks. Iterative pruning to obtain subnetworks [46] or individually pruning subnetworks with the same base network [14] have shown to be effective. Both methods apply a dense-to-sparse training regime causing substantial computational overhead. In the work of [47], two methods with no computational overhead during training and testing were proposed, namely (Efficient) Dynamic Sparse Training Ensembles, or (E)DST Ensembles. For this method, subnetworks are sequentially obtained with the RigL method [22], after an initial exploration phase, by pruning and regrowing a large part (e.g., 80%) of the weights with the smallest magnitude. The authors of [48] showed that the success of a DST method like RigL is likely the result of improved gradient flow in early training by regrowing weights based on high-magnitude gradients, something the Lottery Ticket Hypothesis [4] and SET [20] on their own seem to be less effective at. Despite its efficiency, EDST does not leverage topological sparsity to boost subnetwork diversity, a balance crucial for enhancing ensemble diversity and performance.

2.3 Truly sparse training

Research in the domain of 'truly' sparse training has been limited. This approach involves training neural networks without relying on dense matrix operations filled predominantly with redundant zero-valued weights. Instead, it employs more efficient data representations for enhanced effectiveness. The benefits of using more efficient data representations include improvements in memory overhead as we no longer store redundant zeros and binary masks, as well as an increase in computational efficiency by avoiding unnecessary 0-multiplications. Since the sparsity is inherent in the model's construction rather than being simulated with binary masks, this type of sparsity is referred to as 'native sparsity.' An efficient sparse backpropagation algorithm, *SparseProp*, was introduced in [49], to speed up ANN training on commodity CPUs. They reported speedups in the end-to-end runtime of experiments. Moreover, novel results were presented by [6] and [50] who came up with a new training paradigm called truly sparse training. They implemented the SET algorithm for a regular MLP and a Denoising Autoencoder, allowing an MLP with hundreds of thousands of neurons to be trained on a regular CPU without GPU support. This work was continued by [51] who introduced a parallel training algorithm for truly sparse networks. To the best of our knowledge, the existing methods that use the truly sparse framework do not have the capability to store, re-train, and re-evaluate networks after the cache had been cleared. In this paper, we extend the truly sparse framework to ensemble learning and sequentially train, store, and evaluate, the ensemble of efficient subnetworks.

3 Truly Sparse Ensembles

We present a Truly Sparse Ensemble (TSE)[1] at the intersection of truly sparse training and ensemble learning. Moreover, we propose two algorithms that increase diversity between subnetworks after re-initialization in a sparse ensemble. Our goal is to advance the truly sparse training paradigm by creating an actually sparse ensemble without the overhead of simulating binary masks, while exploiting the topological flexibility of sparse networks to increase the diversity between subnetworks in the ensemble.

3.1 Designing the Truly Sparse Ensemble

We construct an ensemble of truly sparse MLP networks without binary masks. We implement our solution by using the library of [6], which we refer to as the Truly Sparse Framework. Following the Efficient Dynamic Sparse Training (EDST) protocol [47], we sequentially obtain the subnetworks for our ensemble. After the EDST global exploration phase during which we explore the solution space with a large learning rate, we propose a two-stepped *Comprehensive Refinement Phase* (see Sect. 3.2.3) to reach better convergence for the final state of our subnetworks. A new subnetwork is obtained by pruning a large percentage of the weights (e.g., 80%) and randomly regrowing new weights following either of our proposed re-initialization algorithms. To circumvent the need for dense matrix operations, which occur when calculating gradients for all weights in the network, our approach does not rely on gradient magnitude for regrowing weights during the re-initialization of a new subnetwork. Creating a truly sparse implementation with gradient-based regrowing is therefore not ideal as the dense matrices or the computational time required for this (even when the gradients are computed online) reduce the truly sparse advantages. The aim of ensembles in the Truly Sparse Framework is to enable the training of efficient sparse ensembles on resource-limited hardware. The complete ensemble consists of M sparse subnetworks that collectively form the TSE. Similar to [47], the final prediction of the ensemble is given by

$$\hat{y} = argmax(\frac{1}{M}\sum_{i=1}^{M} Softmax([p(a_1^i)...p(a_k^i)] \tag{1}$$

where $\hat{y}$ is the prediction of the ensemble, M the number of subnetworks, and $p(a_k^i)$ the predicted probability of the k^{th} output neuron of the i^{th} subnetwork. We introduce several new functionalities to the truly sparse framework to realize this, including the capability to store networks and retrieve them later. All weight matrices are stored in Compressed Sparse Row format using SciPy [52].

3.2 Subnetwork re-initialization

Subnetwork diversity is of great importance for the performance of an overall ensemble [29–32, 53]. This significance stems from the fact that a lack of diversity among subnetworks often leads to correlated errors; if one subnetwork errors in classification, the others are likely to make similar mistakes [53].

We propose two new algorithms focused on improving subnetwork diversity after re-initialization in an ensemble: Disjoint Truly Sparse Ensemble (Disjoint TSE) and Distance Truly Sparse Ensemble (Distance TSE). The novelty of both methods lies in how we grow new connections when re-initializing a new subnetwork. In contrast to the $EDST_{RG}$ procedure which does not actively diversify subnetworks when regrowing weights, the Disjoint TSE and Distance TSE approaches that we propose adopt distinct strategies for regrowing subnetwork weights. Specifically, Disjoint TSE bases regrowth on topological dissimilarity to all previous subnetworks, while Distance TSE utilizes the Euclidean distance to the preceding subnetwork. Moreover, we introduce a new

[1] Code is available at https://github.com/prdvanderwal/Truly-Sparse-Ensembles

implementation of the refinement phase where we also reduce the frequency with which the weight evolution procedure is applied (evolution frequency).

3.2.1 Disjoint re-initialization

The Disjoint TSE algorithm focuses on improving topological diversity among subnetworks when re-growing weights for the re-initialization of a new subnetwork, inducing structural diversity with a heterogeneous ensemble as a result. We iteratively regrow weights that are not part of the final topology of any previous subnetwork. When the new model is initialized, weights are randomly regrown as long as they are not in the set of "blocked" weights of all converged subnetworks thus far. Given that the time complexity of a look-up in a set is *O(1)*, the computational overhead of this iterative process is marginal. More details on the Disjoint TSE implementation can be found in the pseudocode in Appendix A.1.

The proposed Disjoint TSE algorithm has two variants: (1) Regular Disjoint TSE (RD-TSE) and (2) Fully Disjoint TSE (FD-TSE). For the former variant, the newly grown weights of the new subnetwork are only disjoint with the weight matrices of all previous subnetworks when initialized. During the subsequent training epochs, the network topology of the new subnetwork dynamically adapts and is able to grow weights that are in similar locations as previous subnetworks. Given the high topological sparsity of the subnetworks, we propose a second variant where the regrown weights in the new subnetwork are fully disjoint with any previous subnetwork. An overview of FD-TSE is presented in Fig. 1. The only difference between RD-TSE and FD-TSE is step c. We only apply FD-TSE and RD-TSE to the weight matrices of the hidden layers as all weight matrices are initialized following the Erdős-Rényi distribution as presented by [20]. As a result, for datasets with a relatively low number of input features and output classes, the first and last layers of a subnetwork have too many existing connections

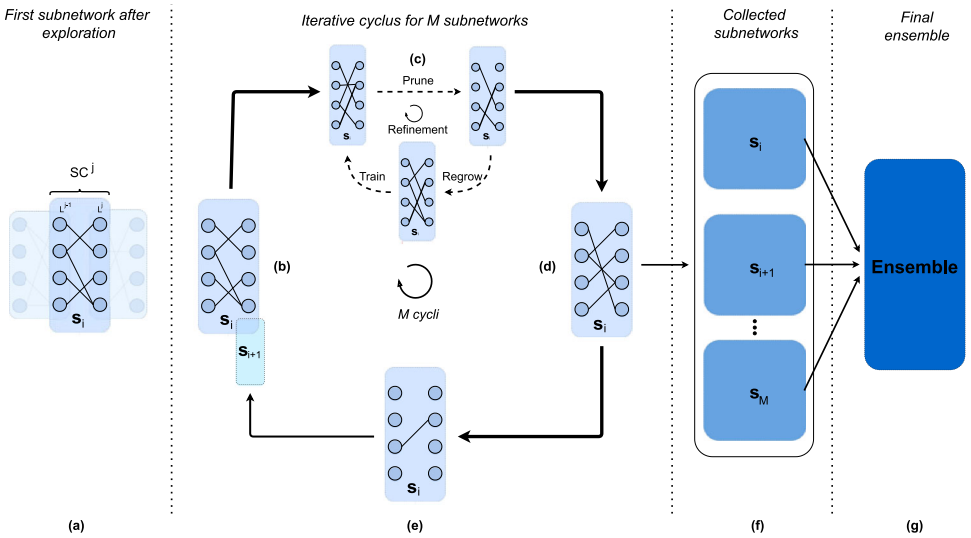


Fig. 1 Schematic overview of the Fully Disjoint TSE (FD-TSE) training procedure. The procedure is visualized for a single sparse connected layer (SC^j) but is applied to all hidden layers. After the initial exploration phase, we obtain the base subnetwork S_i (**a**). We take this network (**b**) and start the refinement phase as depicted in (**c**). The refinement phase entails that for a selected number of epochs, a small fraction of the weights is pruned and regrown where the regrown weights are forced to be disjoint with previous subnetworks in their final states (if applicable). After the refinement phase, we obtain the final state of subnetwork S_i (Spa) which we store for later (**f**). Thicker lines correspond with weights with a higher magnitude. To obtain a new base subnetwork, we prune a very large fraction of the weights that are the smallest in magnitude and obtain the network depicted in (**e**). Subsequently, we regrow the same number of weights disjointly with all previous subnetworks in their final state and obtain the new subnetwork S_{i+1}. The cyclus b-c-d-e is repeated for another $M-1$ times to collect M almost fully disjoint subnetworks which we combine into an ensemble (**g**).

to find disjoint matrices. For consistency, we also apply the Dist-TSE algorithm only to the weight matrices of the hidden layers.

3.2.2 Distance re-initialization

Distance TSE (Dist-TSE) forces the network to regrow the weights so that the Euclidean distance between the weights of the new subnetwork and the preceding subnetwork, exceeds a certain distance within the solution space. Our analysis revealed that the Euclidean distances between the weight matrices of converged subnetworks and the weight matrices of newly initialized subnetworks, follow approximately a Gaussian distribution. Given this property, we use a sampled mean and a factored standard deviation as indicators to find sufficiently distant weight matrices. The pseudocode of Dist-TSE can be found in Appendix A.2.

Our algorithm iteratively finds a sufficiently distant weight matrix for all the hidden layers in the MLP subnetwork. To qualify as the weight matrix for the new subnetwork, a generated weight matrix must exceed the minimum Euclidean distance threshold. This threshold is established by generating 100 random weight matrices and calculating the mean and standard deviation of the Euclidean distances to the preceding network weight matrix. The minimum distance threshold is set by adding the calculated standard deviation, multiplied with the matrix distance coefficient λ (hyperparameter), to the mean. Our empirical analysis revealed that setting the iteration count to 100 offers an optimal balance between computational expense and ensuring a Gaussian-like distribution of Euclidean distances, which is indicative of the running time (see Fig. 2a).

Early testing revealed the boundary values of the coefficient λ after which finding a sufficiently distant matrix quickly becomes intractable. An overview of the impact of the chosen value for the matrix distance coefficient on the running time is presented in Fig. 2b. We observe that the running time strongly increases for values of λ which are larger than 2.0. We, therefore, limited the experiments for the Dist-TSE implementation to λ values of 1.0, 1.5, and 2.0. Nevertheless, a fail-safe was integrated into the algorithm for the unlikely scenario that it is not able to find a fitting matrix.

3.2.3 Refinement phase

The refinement phase as presented by [47] only lowers the learning rate and pruning rate to allow the subnetwork to converge to a more optimal solution. Extending the idea of lowering the learning and pruning rate as done by [22] and [47] to reach better convergence, we propose to also reduce the frequency with which the weight evolution procedure is applied (evolution frequency). The aim of the *Comprehensive Refinement Phase* that we propose is to make the network able to converge better during the refinement phase as it trains for more epochs

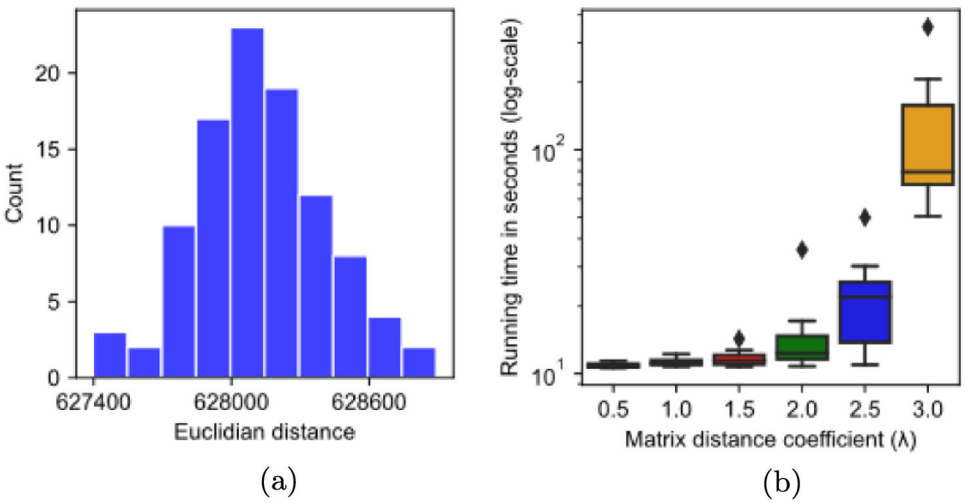


Fig. 2 **a** Euclidean distance distribution after sampling 100 random weight matrices. **b** Boxplot showing how the matrix distance coefficient λ affects the running time to find a distant matrix. For each value of λ, we ran 10 iterations of the distance re-initialization algorithm on the Gesture Phase Segmentation dataset for a 1000×1000 weight matrix at a sparsity level of 0.96.

with the same topology. The intuition behind this is that minimizing the loss during these final steps of training might benefit from being disturbed less as the step size and loss reduction per step are small.

Each subnetwork undergoes a two-step refinement phase after exploration, with equal epochs for both steps. The first step halves the weight evolution frequency, and the second halves the learning rate, pruning rate, and evolution frequency. After a fixed number of epochs (tunable hyperparameter), we obtain the final topology of the subnetwork and enter the cycle of escaping the solution basin by pruning the network with a global pruning rate q, regrowing the weights following either of the implementations above, and resetting the learning rate, pruning rate, and evolution frequency to the values of the first step of the refinement phase and continue training for the next subnetwork. This cycle is repeated M times to cheaply obtain M subnetworks.

4 Evaluation

We demonstrate the effectiveness of the proposed Truly Sparse Ensemble in terms of accuracy, the number of parameters, training floating point operations (FLOPs), and actual training time. We compare our proposed algorithms with several baselines and evaluate the effectiveness of the Comprehensive Refinement Phase with an ablation study.

4.1 Experimental setup

4.1.1 Datasets

We evaluate our methods on 16 datasets, of which 2 are widely used datasets from different domains: the Gesture Phase Segmentation dataset (video segmentation) [54] and HIGGS (physics particles) [55]. The other 14 datasets were selected from the tabular data benchmark[2] [56].

4.1.2 Evaluation metrics

The metrics used to evaluate the performance of the model are the classification accuracy on the test set, the number of weights, the number of training FLOPs, and the actual training time. In contrast to most related work, we do not calculate the number of training FLOPs theoretically for our truly sparse models. Instead, as our implementation allows it, we make use of the Performance Application Programming Interface software [57]. This library provides us with the interface for a hardware performance counter, allowing us to process raw CPU data and extract the actual number of FLOPs performed by the CPU. Moreover, we report the actual training time as we do not make use of binary masking.

4.1.3 Implementation

We constructed all models in the Truly Sparse Framework with three hidden layers, each consisting of 1000 neurons. All ensembles, unless indicated differently, consist of five submodels and were trained for a total of 350 epochs. All layers in each network were initialized with the Erdős-Rényi (ER) distribution as presented by [20]. The hyperparameter ϵ determines the sparsity level S of a model layer. The ER distribution allocates higher sparsity to the layers with more neurons. For most datasets, we used different values of ϵ for our experiments. However, as a result of the aforementioned ER initialization property, the total sparsity level of a model differs slightly per dataset. A model with a 'higher' sparsity level refers to a network with fewer connections. All experiments were run on a single core of a Dell T60 (remote) server with a 2×Silver-4210 processor.

[2] Available at: https://huggingface.co/datasets/inria-soda/tabular-benchmark

4.1.4 Baselines

We consider the EDST model [47] as the baseline for results comparison. However, the EDST model by [47] is implemented in PyTorch [58], a well-established and highly optimized machine learning framework, and makes use of gradient magnitude regrowing. Given the nascency of the Truly Sparse Framework, comparing our models to a model in PyTorch yields a distorted impression of the effectiveness of our algorithms. Instead, we create a new EDST implementation with the Truly Sparse Framework named EDST_{RG}, where RG stands for random growth. We also compare our methods with a single SET-MLP [20] and a single static sparse model. For completeness, we also evaluate fully connected networks as non-sparse baselines, specifically a single dense MLP and an ensemble consisting of dense MLPs (dense ensemble). As we aim to advance performance within the new paradigm of truly sparse training, we implement all these models in the truly sparse framework to allow for an accurate and fair comparison.

4.2 Experimental results

We evaluate the proposed models, Regular Disjoint TSE (RD-TSE), Fully Disjoint TSE (FD-TSE), and Distance TSE (Dist-TSE), against several baselines. An overview of a selection of the experimental results on the Gesture Phase Segmentation dataset and HIGGS dataset can be found in Table 1.

For the two tabular datasets, the Gesture Phase Segmentation dataset and HIGGS, we observe that for almost all variations, the two proposed algorithms consistently outperform the EDST_{RG} implementation and all other baselines without any significant additional overhead in terms of training FLOPS and training time. The inference time of the proposed methods is equal to the baseline models. Interestingly, the best performing method is different per dataset and sparsity level. Dataset characteristics such as the number of features, initialization method, and the number of target classes, most likely affect the effectiveness of our methods on each dataset. Neither varying the number of subnetworks in the ensemble nor the depth or width of the ensemble showed significant differences between the proposed methods. FD-TSE was the only model that benefited from an increase in the number of subnetworks. The results in Table 1 show that the proposed algorithms and variations improve overall ensemble performance on the two selected tabular datasets. To overcome the numerical instability of the Truly Sparse Framework due to its nascency, and evaluate our algorithms at lower levels of sparsity, we implemented both the Regular and Fully Disjoint algorithm in PyTorch [58], the same framework as [47], and found that our algorithms can also be used at lower levels of sparsity.

4.3 Tabular benchmark results

To assess our methods' performance without dataset-specific fine-tuning, we evaluated all models on the numerical classification benchmark for tabular data [56]. More details on the individual datasets can be found in Appendix D. For the training of all models, we used the same configuration as described in Sect. 4.1. For all sparse models, we used an ϵ of 20 for the Erdős-Rényi initialization of the weight matrices. The actual sparsity level differs per dataset as it is dependent on the number of input features as explained in Sect. 4.1. The results of the experiments on the tabular benchmark are presented in Table 2. The best performing model (BPM) metric indicates the number of times a model achieved the highest accuracy on the test set. In 11 out of 14 datasets, one of our methods topped in classification accuracy, most often achieved by Dist-TSE. The disjoint implementations incur minor running time overheads but are still much faster than a single dense model. Table 2 shows that our re-initialization algorithms enhance ensemble performance across various tabular datasets.

Table 1 Summary of experiments of our TSE implementations and baselines on the Gesture Phase Segmentation and HIGGS datasets. We take the single dense model as a reference point for the less intuitive metrics and express the results for the other models as a fraction (...x) of the result of the dense model. For all sparse models, we ran three iterations and report the average classification accuracy and standard deviation. Models marked with * showed statistically significant improvement over the baseline model EDST_{RG} at the corresponding sparsity level in a different experiment. See Appendix B for details

Dataset	Model	Sparsity (S)	Results			
			Accuracy (%)	Weights (#)	Train flops (#)	Train time (min)
Gesture phase	Single dense model	–	54.9	2,058,005	1.71e13	~ 60.0
Segmentation	EDST_{RG}	0.98	71.2 ± 0.4	0.12x	0.04x	~ 19.2 ± 0.4
		0.95	71.8 ± 0.2	0.25x	0.07x	~ 27.9 ± 0.2
		0.93	71.0 ± 0.2	0.37x	0.11x	~ 37.0 ± 0.4
	RD-TSE (ours)	0.98	72.0 ± 0.4	0.12x	0.04x	~ 18.6 ± 0.1
		0.95*	73.1 ± 0.2	0.25x	0.07x	~ 32.3 ± 0.1
		0.93*	73.1 ± 0.3	0.37x	0.11x	~ 55.4 ± 1.3
	FD-TSE (ours)	0.98*	71.9 ± 0.2	0.12x	0.04x	~ 21.5 ± 0.2
		0.95*	73.4 ± 0.7	0.25x	0.07x	~ 40.9 ± 2.8
		0.93*	**74.8 ± 0.3**	0.37x	0.11x	~ 82.1 ± 5.9
	Dist-TSE ($\lambda = 1.5$) (ours)	0.98*	72.7 ± 0.5	0.12x	0.04x	~ 16.7 ± 0.2
		0.95*	72.8 ± 0.1	0.25x	0.07x	~ 24.7 ± 0.2
		0.93*	73.2 ± 0.3	0.37x	0.11x	~ 33.0 ± 0.7
	SET-MLP	0.95	60.6 ± 0.7	0.05x	0.07x	~ 28.3 ± 0.6
	Single static sparse model	0.95	67.2 ± 0.7	0.05x	0.07x	~ 23.5 ± 0.1
	Dense ensemble	–	54.0	5.00x	5.00x	~ 299.9
HIGGS	Single dense model	–	57.1	2,033,002	2.28e14	~ 709.6
	EDST_{RG}	0.98	64.2 ± 0.1	0.12x	0.04x	~ 273.8 ± 10.5
		0.95	63.0 ± 0.1	0.25x	0.07x	~ 397.8 ± 6.1
	RD-TSE (ours)	0.98	**64.7 ± 0.1**	0.12x	0.04x	~ 273.3 ± 2.0
		0.95*	64.0 ± 0.2	0.25x	0.07x	~ 400.6 ± 1.6
	FD-TSE (ours)	0.98*	**64.7 ± 0.1**	0.12x	0.04x	~ 274.0 ± 2.0
		0.95*	64.1 ± 0.3	0.25x	0.07x	~ 403.6 ± 2.5
	Dist-TSE ($\lambda = 1.5$) (ours)	0.98*	**64.7 ± 0.1**	0.12x	0.04x	~ 275.4 ± 3.3
		0.95*	64.0 ± 0.1	0.25x	0.07x	~ 397.3 ± 6.8
	SET-MLP	0.95	54.2 ± 0.2	0.05x	0.07x	~ 381.9 ± 1.0
	Single static sparse model	0.95	53.3 ± 0.6	0.05x	0.07x	~ 378.3 ± 1.7
	Dense ensemble	–	60.4	5.00x	5.00x	~ 3561.5

Bold indicates the best-performing model in terms of classification accuracy

4.4 Ablation study Comprehensive Refinement Phase

Next to our two proposed algorithms, we also suggest reducing the frequency with which we apply the topology update (evolution frequency) during the refinement phase. We hypothesized that this would allow the network to converge better. In this section, we present the results of the ablation study on the impact of this change in the refinement phase. Specifically, we compare the refinement phase as discussed by [47] where only the learning rate and pruning rate are halved during the two-stepped refinement phase, to our proposed *Comprehensive Refinement Phase*. The results of the ablation study are reported in Table 3.

In line with our hypothesis, we observe that for the tabular datasets, our proposed refinement phase is predominantly beneficial. We did not observe any significant difference in terms of FLOPs and running time for the ablation experiments.

Table 2 Aggregated results of experiments on 14 numerical classification datasets from the tabular data benchmark [56]. The best performing model (BPM) metric represents the frequency that a model got the highest classification accuracy of all models

Dataset	Model	BPM [#]	Total train time [min]
Aggregated datasets	Single dense model	0	$\sim$ 2011
	EDST_{RG}	2	$\sim$ 827
	RD-TSE (ours)	2.33	$\sim$ 899
	FD-TSE (ours)	2.33	$\sim$ 1010
	Dist-TSE ($\lambda = 1.5$) (ours)	**6.33**	$\sim$ 823
	SET-MLP	1	$\sim$ 820
	Single static sparse model	0	$\sim$ 789
	Dense ensemble	–	–

Bold indicates the best-performing model in terms of BPM metric

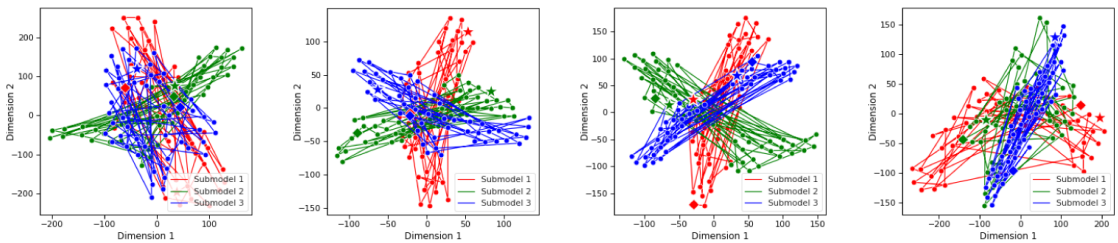
Table 3 Ablation study on the impact of the Comprehensive Refinement Phase on the Gesture Phase Segmentation (GPS) and HIGGS datasets

Model	Comprehensive Refinement	Test accuracy [%]	
		GPS	HIGGS
RD-TSE	No	71.2	**63.7**
	Yes	**72.6**	63.5
FD-TSE	No	73.9	63.5
	Yes	**74.2**	**63.7**
Dist-TSE ($\lambda = 1.5$)	No	71.7	63.5
	Yes	**73.4**	**63.5**

Bold indicates the best-performing model in terms of classification accuracy

4.5 Training trajectory

We use t-SNE [59] to visualize the training trajectories of our submodels in the solution space. We project the output of the Softmax layer computed on the samples of the test set onto a 2D space. Figure 3 gives an overview of the different training trajectories of three subnetworks on the Gesture Phase Segmentation dataset. We observe that the trajectories of the subnetworks of the EDST_{RG} implementation (3a) and Dist-TSE (3d) seem fairly random and sometimes have shared trajectories. This was to be expected as there is no regularization in place for the models to take a different trajectory direction besides random re-initialization (in terms of topological



(a) EDST_{RG} (b) RD-TSE (c) FD-TSE (d) Dist-TSE

Fig. 3 t-SNE projection of the training trajectories of three subnetworks discovered by various TSE implementations on the GPS dataset. For Dist-TSE, we set $\lambda = 1.5$. The diamonds and stars represent the start and end of each subnetwork, respectively. The sparsity level is $S = 0.95$.

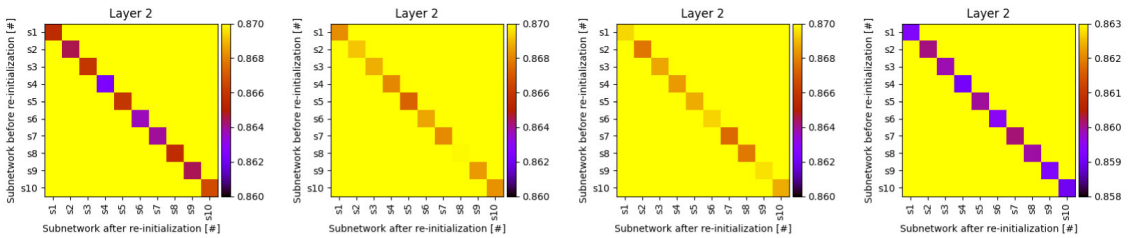
similarity). For Fig. 3b, we see that the training trajectory of a new subnetwork is different from that of the previous networks and that the model is mostly preserving this difference in direction. In Fig. 3c, the quasi-orthogonal trajectories across each subnetwork's refinement phases suggest the algorithm's effectiveness in achieving disjoint networks. However, the point distances in t-SNE, subject to potential information loss due to dimensionality reduction, do not provide significant insights, as global distances are not reliably preserved.

4.6 Topological distances between subnetworks

Building on the findings of [60], we visualize the topological distances between subnetworks for each proposed method using NNSTD [60]. This metric accounts for the isomorphic nature of ANNs. We evaluate the effectiveness of our proposed algorithms by visualizing the topological distance between a converged subnetwork and the succeeding subnetwork that has just been re-initialized with one of our algorithms. For the EDST_{RG} implementation and for each of our proposed methods, we train 10 independent subnetworks for a duration of one exploration phase and one refinement phase. We subsequently store the converged topologies and re-initialize all subnetworks with one of our methods. In Fig. 4, a visualization of the topological distances between the converged subnetworks and re-initialized subnetworks is presented. We observe from Fig. 4b and c that disjoint re-initialization yields a more distant subnetwork topology from its predecessor compared to regrowing the weights without considering subnetwork diversity as is done in the EDST_{RG} (Fig. 4a). We surprisingly also see that the Distance TSE algorithm (Fig. 4d) consistently yields topologies that are closer to each other compared to the EDST_{RG} implementation. A possible explanation could be that an effective way of finding a distant matrix would be for the new subnetwork to have a negative weight in the same location where its predecessor had a positive weight and vice-versa. As the resulting Euclidean distance is higher for this instance than for regrowing a weight somewhere the subnetwork's predecessor did not have a weight, our iterative process of finding a distance matrix might unintentionally give a slight preference for subnetworks with a more similar topology.

4.7 Scalability

To get a better understanding of how our proposed algorithms scale with an increasing number of neurons, we conducted some experiments for each of our proposed methods with a larger number of neurons. More specifically, we ran 10 iterations of each of our proposed re-initialization algorithms for a weight matrix of two hidden layers with a variable number of neurons. We used $\epsilon = 10$ to control the sparsity level and $\lambda = 1.5$ for Dist-TSE. The results are presented in Table 4.



(a) EDST_{RG} (b) RD-TSE (c) FD-TSE (d) Dist-TSE

Fig. 4 Heatmap representing the topological distances measured by the NNSTD method [60] between the second layer of thoroughly trained subnetworks (x-axis) and the second layer of subnetworks that were obtained directly following re-initialization (y-axis) using various TSE implementations on the Gesture Phase Segmentation dataset. For Dist-TSE we set $\lambda = 1.5$. The parameter $w_i (i = 0, 1, ..., 9)$ represents 10 independently trained and evaluated subnetworks. Note, the heatmap in (d) has a different color scale. A topology similarity value of 0 means that the networks are identical, while a value of 1 means that the models' topologies are completely different. The sparsity level is $S = 0.95$.

Table 4 Summary of results of all experiments evaluating the scalability of our proposed method. We ran 10 iterations for each method and report the average running time with standard deviation. For D-EDST, we also report the average number of iterations it took to find a sufficiently distant matrix. We used a value of $\lambda = 1.5$ for D-EDST and set the sparsity parameter $\epsilon = 10$ for all experiments

Model	Weight matrix size	Avg. running time (sec.)	Avg. iterations
RD-TSE	1,000×1,000	0.081 ± 0.02	–
	5,000×5,000	0.387 ± 0.03	–
	10,000×10,000	0.813 ± 0.08	–
	15,000×15,000	1.120 ± 0.09	–
FD-TSE	1,000×1,000	0.082 ± 0.03	–
	5,000×5,000	0.732 ± 0.16	–
	10,000×10,000	0.723 ± 0.07	–
	15,000×15,000	1.051 ± 0.05	–
Dist-TSE	1,000×1,000	11.65 ± 1.03	109.8 ± 9.54
	5,000×5,000	50.11 ± 8.47	126.9 ± 17.2
	10,000×10,000	153.4 ± 16.9	116.7 ± 13.4
	15,000×15,000	278.2 ± 19.9	109.2 ± 6.70

We observe that both RD-TSE and FD-TSE scale well with the number of neurons, showing negligible increases in running time. While Dist-TSE's running time does increase with more neurons, the average number of iterations (minimum = 100) remains constant for larger matrices. From this, it can be derived that the main cause for extra running time is caused by the 100 iterations which is used to set a minimum distance and not by the increased difficulty of finding a sufficiently distant matrix.

4.8 Limitations

This work introduces a new perspective on subnetwork diversification in dynamic sparse ensembles by introduction of two new algorithms for ensemble diversification, along with a software contribution and the Truly Sparse Ensemble. The primary challenge we faced is the difficulty in quantitatively comparing our results with existing literature, as almost all related methods rely on dense matrices. In contrast, our approach employs truly sparse training, which offers significant computational efficiency that dense frameworks cannot match.

Although some configurations showed numerical instability, this highlights the need for software improvements rather than a limitation of the method itself. Our results demonstrated the effectiveness of our algorithms but did so for both basic sparsity and ensemble training methods. Future work could focus on integrating more advanced methods like boosting, exploring new regrowth criteria such as gradient magnitude regrowth [22, 47, 48], and extending the approach to different architectures to enhance both performance and stability.

5 Conclusions

We proposed two algorithms to improve diversity among subnetworks in ensembles in a truly sparse context, with novel strategies to grow new connections when re-initializing a new subnetwork. At the intersection of truly sparse training and ensemble learning, we have effectively introduced a novel concept: the Truly Sparse Ensemble. We moved beyond the typical approach of simulating sparsity with binary masks, and implemented our proposed algorithms in a truly sparse manner to take full benefits of unstructured sparsity during training. We evaluated the learning capabilities of the Truly Sparse Ensembles on various datasets. Our methods surpass both sparse and dense baseline models in performance on tabular data, achieving this with minimal computational overhead in terms of floating point operations and running time. Moreover, all training was conducted using a single CPU core, illustrating that truly sparse models can be efficiently trained on commodity hardware without significantly prolonged training times, exemplifying the potential of the truly sparse training paradigm. Our

proposed sparse models notably exceed the performance of dense baseline models in both accuracy and running time.

Algorithms

Disjoint re-initialization algorithm

Algorithm 1 Truly Sparse Ensemble with disjoint re-initialization

Data: Layer i to k with: Sparse Weight Matrix $\mathbf{W}_i$, prune rate p, global exploration rate q, and evolution frequency e

%Initialization ;
model ← initialize Truly Sparse MLP (TS-MLP) ;
for $i \leftarrow 0$ **to** k **do**
    $\boldsymbol{W}_i \leftarrow$ ErdosRenyiInit($\boldsymbol{W}_i$) ;
    $\boldsymbol{b}_i \leftarrow$ ZeroVector() ;
    $\mathbf{Blocked}_i \leftarrow \varnothing$;
end
globalExploration ← CalculateGlobalPruningEpochs(*numEpochs*) ;
%Training ;
for *epoch* ← 0 **to** *numEpochs* **do**
    model.train_step() ;
    if *epoch* % $e == 0$ **then**
        model.RigL_weight_evolution(p) ;
    if *epoch* in globalExploration **then**
        for $i \leftarrow 0$ **to** $k - 1$ **do**
            if $i == 0$ **then**
                $\boldsymbol{W}_i$.RigL_weight_evolution(q) ;
            else
                for $j \leftarrow 0$ **to** $\boldsymbol{W}_i$.numRows() **do**
                    $\mathbf{Blocked}_i$.add(tuple($\boldsymbol{W}_i$.row[j] , $\boldsymbol{W}_i$.col[j])) ;
                end
                $\boldsymbol{W}'_i \leftarrow$ PruneSmallestK($|\boldsymbol{W}_i|$, q) ;
                numWeightsRegrown ← 0 ;
                iteration ← 0 ;
                while numWeightsRegrown $\neq K$ **do**
                    proposedWeights ← (RegrowRandomK(q) - numWeightsRegrown) ;
                    for $j \leftarrow 0$ **to** $\boldsymbol{W}_i$.numRows() **do**
                        if *iteration* > *maxIterations* **then**
                            store(proposedWeights.numRows()) ;
                            break
                        else if tuple(proposedWeights.row[j], proposedWeights.col[j]) in $\mathbf{Blocked}_i$ **then**
                            proposedWeights.delete(j) ;
                    end
                    $\boldsymbol{W}'_i \leftarrow \boldsymbol{W}'_i \cup$ proposedWeights ;
                    numWeightsRegrown += proposedWeights.numRows() ;
                    iteration += 1 ;
                end
                $\boldsymbol{W}_i \leftarrow \boldsymbol{W}'_i \cup$ proposedWeights ;
        end
end

Distance re-initialization algorithm

Algorithm 2 Truly Sparse Ensemble with distance re-initialization

Data: Layer i to k with: Sparse Weight Matrix $\mathbf{W}_i$, prune rate p, global exploration rate q, evolution frequency e, and matrix distance coefficient λ

%Initialization
model ← initialize Truly Sparse MLP (TS-MLP);
for $i \leftarrow 0$ **to** k **do**
  $\boldsymbol{W}_i \leftarrow$ ErdosRenyiInit($\boldsymbol{W}_i$) ;
  $\boldsymbol{b}_i \leftarrow$ ZeroVector() ;
end
globalExploration ← CalculateGlobalPruningEpochs(*numEpochs*)
%Training;
for *epoch* ← 0 **to** *numEpochs* **do**
  model.train_step() **if** *epoch* % e == 0 **then**
    model.RigL_weight_evolution(p) ;
  if *epoch* in globalExploration **then**
    for $i \leftarrow 0$ **to** $k-1$ **do**
      if i == *0* **then**
        $\boldsymbol{W}_i$.RigL_weight_evolution(q);
      else
        $\boldsymbol{W}'_i \leftarrow$ PruneSmallestK($|\boldsymbol{W}_i|$, q) ;
        for $l \leftarrow 0$ **to** *100* **do**
          totalDistance += CalculateDistance($\boldsymbol{W}_i$, $\boldsymbol{W}'_i \cup$ RegrowRandomK(q))
        end
        $\mu \leftarrow$ GetAverage(totalDistance) ;
        $\sigma \leftarrow$ GetStandardDeviation(totalDistance) ;
        currentDistance ← 0 ;
        iteration ← 0 ;
        while *currentDistance* < ($\mu + \lambda \times \sigma$) **do**
          if *iteration* > *maxIterations* **then**
            Throw Iteration Error and stop training ;
          proposedWeights ← $\boldsymbol{W}'_i \cup$ RegrowRandomK(q) ;
          currentDistance ← CalculateDistance($\boldsymbol{W}_i$, proposedWeights) ;
          iteration += 1 ;
        end
        $\boldsymbol{W}_i \leftarrow \boldsymbol{W}'_i \cup$ proposedWeights ;
    end
end

Statistical analysis

In order to test the statistical significance of the increase in performance of our models presented in Table 1 compared to the sparse baseline EDST_{RG} model, we conduct a paired statistical test for the two datasets in Table 1: the Gesture Phase Segmentation dataset and HIGGS. We bootstrap the test set of the respective datasets 30 times and do a paired comparison between the ensemble accuracies of the sparse baseline model EDST_{RG} and either of our proposed methods FD-TSE, RD-TSE, and Dist-TSE. Through the use of Shapiro–Wilkinson tests we discover that normality cannot be assumed for the distribution of ensemble accuracies. Therefore, we conducted the one-tailed nonparametric Wilcoxon test to verify that each of our proposed models outperforms the baseline model at each sparsity level. The results are presented in Table 5.

With the exception of two models (†), all proposed models significantly outperform the baseline at $p = 0.05$.

Table 5 Statistical evaluation of ensemble accuracies on 30 bootstrapped test sets per sparsity level

D	Sparsity	Model	Acc. (%)	Wilcoxon	p
Gesture Phase Segmentation	0.98	$EDST_{RG}$	70.5 ± 1.1	–	–
		RD-TSE (ours) †	70.4 ± 0.9	300	0.92
		FD-TSE (ours)	70.7 ± 1.0	115	2.2e−2
		Dist-TSE ($\lambda = 1.5$) (ours)	71.0 ± 1.1	46	1.7e−4
	0.95	$EDST_{RG}$	70.8 ± 0.9	–	–
		RD-TSE (ours)	72.2 ± 0.9	0	8.6e−7
		FD-TSE (ours)	71.4 ± 1.0	14	5.0e−6
		Dist-TSE ($\lambda = 1.5$) (ours)	71.9 ± 1.1	0	1.9e−6
	0.93	$EDST_{RG}$	70.3 ± 0.8	–	–
		RD-TSE (ours)	72.0 ± 0.7	0	8.2e−7
		FD-TSE (ours)	72.5 ± 0.9	0	8.4e−7
		Dist-TSE ($\lambda = 1.5$) (ours)	72.2 ± 0.8	0	8.5e−7
HIGGS	0.98	$EDST_{RG}$	64.2 ± 0.2	–	–
		RD-TSE (ours) †	64.1 ± 0.2	202	0.99
		FD-TSE (ours)	64.2 ± 0.2	51	3.6e−2
		Dist-TSE ($\lambda = 1.5$) (ours)	64.3 ± 0.2	51	1.9e−2
	0.95	$EDST_{RG}$	63.2 ± 0.2	–	–
		RD-TSE (ours)	63.5 ± 0.2	0	1.6e−6
		FD-TSE (ours)	63.7 ± 0.2	0	7.8e−7
		Dist-TSE ($\lambda = 1.5$) (ours)	63.6 ± 0.2	0	1.2e−6

For each sparsity level, all models were evaluated on the same sets, and Wilcoxon statistics were used to compare accuracy distributions between the baseline model, $EDST_{RG}$, and our proposed models

Hyperparameters

Given the very limited availability of the literature on the implementation of the Truly Sparse Framework, we mostly used the same configuration of hyperparameters as presented by [51]. For all training, we used gradient descent with a momentum of 0.9 and a weight decay of 0.0002. In contrast to the works of [51] and [20], we used a slightly lower dropout rate of 0.2 as our experiments concerned very sparse networks. We maintained the inclusion of dropout in our training process, following the findings of [51], who demonstrated that integrating dropout significantly aids in preventing overfitting. An overview of the main hyperparameters is given in Table 6.

Moreover, we used the Alternated Left ReLU activation function as presented in [51] for all layers except the last layer to which we applied the Softmax activation function. For the Gesture Phase Segmentation dataset, no value for the slope of the negative side of the input of the Alternated Left ReLU (α) was available in the literature. We thus selected a neutral value of 0.5. For the aggregated datasets we instead applied the regular ReLU activation function as finding a single value for α that suits all 14 datasets was highly unlikely, given the results presented in [51] on the tuning of this hyperparameter. All dense models were also trained with the ReLU activation function as All-ReLU was specifically designed for sparse networks. All models were trained with a batch size of 128. The learning rate η and evolution frequency e varied for the TSE ensembles as a result of the refinement phases. Here, the evolution frequency refers to how often, once every e epochs, we do a topology update. For the dense models, we initially ran all experiments with a fixed learning rate of 0.01 as presented by [20]. We decreased this learning rate to 0.001 for the HIGGS and the Gesture Phase Segmentation datasets. All weights were initialized with He Initialization.

Table 6 Table of hyperparameters used for the experiments

Experiment	Dataset	Architecture	Hyperparameters			
			η	e	$\mathcal{A}$	α
Truly Sparse Ensembles	GPS	50×1000×1000×1000×5	0.1−0.05	2-4-8	All-ReLU	0.5
	HIGGS	28×1000×1000×1000×2	0.1−0.05	2-4-8	All-ReLU	0.05
	Tab. Benchmark	...×1000×1000×1000×2	0.1−0.05	2-4-8	ReLU	–
Baseline ensemble	GPS	50×1000×1000×1000×5	0.01	2	All-ReLU	0.5
	HIGGS	28×1000×1000×1000×2	0.01	2	All-ReLU	0.05
	Tab. Benchmark	...×1000×1000×1000×2	0.01	2	ReLU	–
Dense models	GPS	50×1000×1000×1000×5	0.001	–	ReLU	–
	HIGGS	28×1000×1000×1000×2	0.001	–	ReLU	–
	Tab. Benchmark	...×1000×1000×1000×2	0.001	–	ReLU	–

Table 7 Summary of datasets contained in the tabular benchmark

Dataset	Dataset properties	
	Features	Samples [#]
Bank marketing	7	10578
Bioresponse	419	3434
California	8	20634
Credit	10	16714
Default-of-credit-card-clients	20	13272
Diabetes130US	7	71090
Electricity	7	38474
Eye Movements	20	7608
Heloc	22	10000
House16	16	13488
Jannis	54	57580
MagicTelescope	10	13376
MiniBooNE	50	72998
Pol	26	10082

Tabular benchmark datasets

For all datasets, an 80-20 train split was used to obtain separate sets for training and testing. An overview of the data properties for all datasets can be found in Table 7. All datasets were transformed to binary classification problems by [56]. The HIGGS dataset was already part of the main experiments and was excluded from this benchmark to prevent duplicate experiments.

Acknowledgements The authors would like to acknowledge the financial support of the CogniGron research center and the Ubbo Emmius Funds (Univ. of Groningen).

Author contributions Peter R.D. van der Wal conceptualized the study, conducted the experiments, and wrote the initial and final draft of the manuscript. Nicola Strisciuglio provided guidance throughout the research process, contributed to the interpretation of results, and participated in writing and revising the manuscript. George Azzopardi contributed to the refinement of the analysis and participated in the writing and revision of the manuscript. Decebal Constatin Mocanu provided guidance throughout the research process, contributed to the interpretation of results, and reviewed the manuscript for critical insights.

Funding This work was partially funded by the CogniGron research center and the Ubbo Emmius Funds (Univ. of Groningen).

Data availability All datasets used for this study are publicly available datasets. The Gesture Phase Segmentation dataset is available at https://doi.org/10.24432/C5Z32C. The HIGGS dataset is available at https://doi.org/10.24432/C5V312. The tabular benchmark datasets for classification are available at https://huggingface.co/datasets/inria-soda/tabular-benchmark.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Dong X, Yu Z, Cao W, Shi Y, Ma Q (2020) A survey on ensemble learning. Front Comp Sci 14(2):241–258. https://doi.org/10.1007/s11704-019-8208-z
2. Gale T, Elsen E, Hooker S (2019) The state of sparsity in deep neural networks. CoRR **abs/1902.09574**. arXiv:1902.09574
3. Herculano-Houzel S, Mota B, Wong P, Kaas JH (2010) Connectivity-driven white matter scaling and folding in primate cerebral cortex. Proc Natl Acad Sci 107(44):19008–19013
4. Frankle J, Carbin M (2018) *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. https://openreview.net/forum?id=rJl-b3RcF7
5. Sokar G, Atashgahi Z, Pechenizkiy M, Mocanu DC (2022) *Where to Pay Attention in Sparse Training for Feature Selection?*. https://openreview.net/forum?id=xWvI9z37Xd
6. Liu S, Mocanu DC, Matavalam ARR, Pei Y, Pechenizkiy M (2021) Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. Neural Comput Appl 33(7):2589–2604. https://doi.org/10.1007/s00521-020-05136-7
7. Putra M, Hermawan A, Kim DS, Lee JM (2023) Data prediction-based energy-efficient architecture for industrial IoT. IEEE Sens J 23(14):15856–15866. https://doi.org/10.1109/JSEN.2023.3280485
8. Dix J, Holleman J, Blalock B (2023) Programmable energy-efficient analog multilayer perceptron architecture suitable for future expansion to hardware accelerators. J Low Power Electron Appl. https://doi.org/10.3390/jlpea13030047
9. Mocanu DC, Mocanu E, Nguyen PH, Gibescu M, Liotta A (2016) A topological insight into restricted Boltzmann machines. Mach Learn 104(2–3):243–270
10. Liu B, Wang M, Foroosh H, Tappen M, Pensky M (2015) *Sparse convolutional neural networks*, In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 806–814
11. Bibikar S, Vikalo H, Wang Z, Chen X (2022) Federated dynamic sparse training: Computing less, communicating less, yet learning better. Proceedings of the AAAI Conference on Artificial Intelligence 36(6):6080–6088. https://doi.org/10.1609/aaai.v36i6.20555
12. Liu J, Xu Z, Shi R, Cheung RCC, So HK (2020) *Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers*, in *International Conference on Learning Representations*. https://openreview.net/forum?id=SJlbGJrtDB
13. Wu B, Yu T, Chen K, Liu W (2024) Edge-side fine-grained sparse cnn accelerator with efficient dynamic pruning scheme. IEEE Transactions on Circuits and Systems I: Regular Papers pp. 1–14. https://doi.org/10.1109/TCSI.2023.3347417
14. Whitaker T, Whitley D (2022) Prune and tune ensembles: low-cost ensemble learning with sparse independent subnetworks. In Proceedings of the AAAI Conference on Artificial Intelligence 36:8638–8646
15. Cho M, Joshi A, Hegde C (2021) *ESPN: Extremely Sparse Pruned Networks*, In *2021 IEEE Data Science and Learning Workshop (DSLW)*, pp. 1–8. https://doi.org/10.1109/DSLW51110.2021.9523404

16. Karnin E (1990) A simple procedure for pruning back-propagation trained neural networks. IEEE Trans Neural Networks 1(2):239–242. https://doi.org/10.1109/72.80236. (**Conference Name: IEEE Transactions on Neural Networks**)
17. LeCun Y, Denker J, Solla S (1989) *Optimal Brain Damage*, in *Advances in Neural Information Processing Systems*, vol. 2, ed. by D. Touretzky (Morgan-Kaufmann, 1989). https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf
18. Zhu M, Gupta S (2017) To prune, or not to prune: exploring the efficacy of pruning for model compression
19. Dey S, Huang KW, Beerel PA, Chugg KM (2019) Pre-defined sparse neural networks with hardware acceleration. IEEE J Emerg Select Topics Circuits Syst 9(2):332–345
20. Mocanu DC, Mocanu E, Stone P, Nguyen PH, Gibescu M, Liotta A (2018) Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat Commun 9(1):1–12
21. Liu S, Tian Y, Chen T, Shen L (2023) Don't be so dense: sparse-to-sparse Gan training without sacrificing performance. Int J Comput Vision 131(10):2635–2648. https://doi.org/10.1007/s11263-023-01824-8
22. Evci U, Gale T, Menick J, Castro PS, Elsen E (2020) *Rigging the lottery: Making all tickets winners*, in *International Conference on Machine Learning* (PMLR, 2020), pp. 2943–2952
23. Liu S, Ni'Mah I, Menkovski V, Mocanu DC, Pechenizkiy M (2021) Efficient and effective training of sparse recurrent neural networks. Neural Computing and Applications 33(15):9625–9636. https://doi.org/10.1007/s00521-021-05727-yhttps://rdcu.be/cegM5
24. Dettmers T, Zettlemoyer L (2019) Sparse networks from scratch: Faster training without losing performance. CoRR **abs/1907.04840**. arXiv:1907.04840
25. Yuan G, Ma X, Niu W, Li Z, Kong Z, Liu N, Gong Y, Zhan Z, He C, Jin Q et al (2021) Mest: accurate and fast memory-economic sparse training framework on the edge. Adv Neural Inf Process Syst 34:20838–20850
26. Mostafa H, Wang X (2019) *Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization*, In *International Conference on Machine Learning* (PMLR, 2019), pp. 4646–4655
27. Ganaie MA, Hu M, Malik AK, Tanveer M, Suganthan PN (2022) Ensemble deep learning: a review. Eng Appl Artif Intell 115:105151
28. Zhou ZH (2025) Ensemble methods: foundations and algorithms. CRC press
29. Ren Y, Zhang L, Suganthan P (2016) Ensemble classification and regression-recent developments, applications and future directions. IEEE Comput Intell Mag 11(1):41–53. https://doi.org/10.1109/MCI.2015.2471235. (**Conference Name: IEEE Computational Intelligence Magazine**)
30. Sagi O, Rokach L (2018) Ensemble learning: A survey. WIREs Data Mining and Knowledge Discovery 8(4):e1249. https://doi.org/10.1002/widm.1249https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1249
31. Chandra A, Yao X (2004)*DIVACE: Diverse and Accurate Ensemble Learning Algorithm*, in *Intelligent Data Engineering and Automated Learning – IDEAL 2004*, ed. by Z.R. Yang, H. Yin, R.M. Everson (Springer, Berlin, Heidelberg), Lecture Notes in Computer Science, pp. 619–625. https://doi.org/10.1007/978-3-540-28651-6_91
32. Chen S, Zhao R, Fu H (2021) *Ensemble diversity enhancement based on parameters evolution of base learners*, in *2021 33rd Chinese Control and Decision Conference (CCDC)*, pp. 5887–5893. https://doi.org/10.1109/CCDC52312.2021.9601781. ISSN: 1948-9447
33. Liu Y, Yao X (1999) Ensemble learning via negative correlation. Neural Networks 12(10):1399–1404. https://doi.org/10.1016/S0893-6080(99)00073-8https://www.sciencedirect.com/science/article/pii/S0893608099000738
34. Moreira J, Soares C, Jorge A, Sousa J (2012) Ensemble approaches for regression: a survey. ACM Comput Surv 45:10:1-10:40. https://doi.org/10.1145/2379776.2379786
35. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140. https://doi.org/10.1007/BF00058655
36. Webb G, Zheng Z (2004) Multistrategy Ensemble Learning: Reducing Error by Combining Ensemble Learning Techniques. Knowledge and Data Engineering, IEEE Transactions on 16:980–991. https://doi.org/10.1109/TKDE.2004.29
37. Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844. https://doi.org/10.1109/34.709601. (**Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence**)
38. Ho TK (1995) Random decision forests. In Proceedings of 3rd international conference on document analysis and recognition 1:278–282 (**IEEE**)
39. Freund Y, Schapire R (1996) Experiments with a New Boosting Algorithm. undefined. https://api.semanticscholar.org/CorpusID:1836349
40. Freund Y, Schapire RE (1997) A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences 55(1):119–139. https://doi.org/10.1006/jcss.1997.1504https://www.sciencedirect.com/science/article/pii/S002200009791504X
41. Wolpert DH (1992) Stacked generalization. Neural Networks 5(2):241–259. https://doi.org/10.1016/S0893-6080(05)80023-1https://www.sciencedirect.com/science/article/pii/S0893608005800231
42. Ciregan D, Meier U, Schmidhuber J (2012) *Multi-column deep neural networks for image classification*, in *2012 IEEE conference on computer vision and pattern recognition* (IEEE), pp. 3642–3649

43. Harangi B, Baran A, Beregi-Kovacs M, Hajdu A (2023) Composing diverse ensembles of convolutional neural networks by penalization. Mathematics 11(23). https://doi.org/10.3390/math11234730https://www.mdpi.com/2227-7390/11/23/4730
44. Lee S, Purushwalkam S, Cogswell M, Crandall D, Batra D (2015) Why m heads are better than one: Training a diverse ensemble of deep networks. arXiv preprint arXiv:1511.06314
45. Abe T, Buchanan EK, Pleiss G, Cunningham JP (2023) Pathologies of predictive diversity in deep ensembles. arXiv preprint arXiv:2302.00704
46. Kobayashi S, Kiyono S, Suzuki J, Inui K (2022) *Diverse Lottery Tickets Boost Ensemble from a Single Pretrained Model*, in *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models* (Association for Computational Linguistics, virtual+Dublin, 2022), pp. 42–50. https://doi.org/10.18653/v1/2022.bigscience-1.4. https://aclanthology.org/2022.bigscience-1.4
47. Liu S, Chen T, Atashgahi Z, Chen X, Sokar G, Mocanu E, Pechenizkiy M, Wang Z, Mocanu DC (2022) *Deep Ensembling with No Overhead for either Training or Testing: The All-Round Blessings of Dynamic Sparsity*, in *ICLR*
48. Evci U, Ioannou Y, Keskin C, Dauphin Y (2022) Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win. Proceedings of the AAAI Conference on Artificial Intelligence 36(6):6577–6586. https://doi.org/10.1609/aaai.v36i6.20611https://ojs.aaai.org/index.php/AAAI/article/view/20611. Number: 6
49. Nikdan M, Pegolotti T, Iofinova E, Kurtic E, Alistarh D (2023) *SparseProp: Efficient Sparse Backpropagation for Faster Training of Neural Networks at the Edge*, in *Proceedings of the 40th International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, vol. 202, ed. by A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, J. Scarlett (PMLR, 2023), pp. 26215–26227. https://proceedings.mlr.press/v202/nikdan23a.html
50. Atashgahi Z, Sokar G, van der Lee T, Mocanu E, Mocanu DC, Veldhuis R, Pechenizkiy M (2022) Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. Machine Learning pp. 1–38
51. Curci S, Mocanu DC, Pechenizkiyi M (2021) Truly sparse neural networks at scale. arXiv preprint arXiv:2102.01732
52. ...Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P (2020) SciPy 1.0 contributors, sciPy 1.0: fundamental algorithms for scientific computing in python. Nat Methods 17:261–272. https://doi.org/10.1038/s41592-019-0686-2
53. Dietterich TG (2000) *Ensemble Methods in Machine Learning*, in *Multiple Classifier Systems* (Springer, Berlin, Heidelberg), Lecture Notes in Computer Science, pp. 1–15. https://doi.org/10.1007/3-540-45014-9_1
54. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
55. Baldi P, Sadowski P, Whiteson D (2014) Searching for exotic particles in high-energy physics with deep learning. Nature Communications 5:4308 https://archive.ics.uci.edu/ml/datasets/HIGGS
56. Grinsztajn L, Oyallon E, Varoquaux G (2022) *Why do tree-based models still outperform deep learning on typical tabular data?*. https://openreview.net/forum?id=Fp7__phQszn
57. Terpstra D, Jagode H, You H, Dongarra J (2010) Collecting Performance Data with PAPI-C. In: Müller MS, Resch MM, Schulz A, Nagel WE (eds) Tools for high performance computing 2009. Springer, Heidelberg, pp 157–173
58. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch
59. Maaten Lvd, Hinton G (2008) Visualizing Data using t-SNE. Journal of Machine Learning Research **9**(86), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html
60. Liu S, der Lee TV, Yaman A, Atashgahi Z, Ferraro D, Sokar G, Pechenizkiy M, Mocanu DC (2020) *Topological insights into sparse neural networks*, in *Joint European conference on machine learning and knowledge discovery in databases* (Springer), pp. 279–294

Authors and Affiliations

Peter R. D. van der Wal[1] · Nicola Strisciuglio[2] · George Azzopardi[1] · Decebal Constatin Mocanu[3]

✉ Peter R. D. van der Wal
p.r.d.van.der.wal@rug.nl

Nicola Strisciuglio
n.strisciuglio@utwente.nl

George Azzopardi
g.azzopardi@rug.nl

Decebal Constatin Mocanu
decebal.mocanu@uni.lu

[1] Information Systems Group, Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

[2] Data Management and Biometrics Group, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Hallenweg 19, 7522 NH Enschede, The Netherlands

[3] Department of Computer Science, Faculty of Science, Technology, and Medicine, University of Luxembourg, Maison du Nombre, 6, avenue de la Fonte, 4364 Esch, Luxembourg