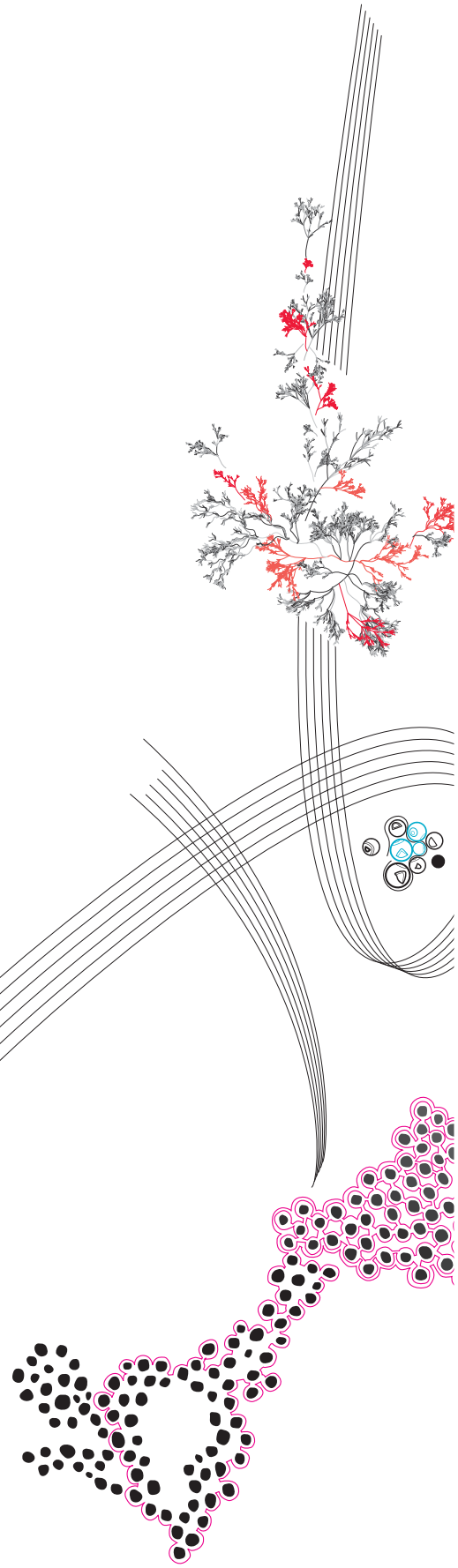


MSc. Thesis Computer Science

Diversifying Multilayer Perceptron Ensembles in a Truly Sparse Context

Peter R.D. van der Wal

Supervisors
dr. Nicola Strisciuglio (committee chair)
dr. Decebal C. Mocanu
dr. Yanqiu Huang

March 10, 2023

Data Management & Biometrics Group
Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

***Abstract*—Artificial Neural Networks are state-of-the-art machine learning models, outperforming their competitors in many fields. One of the major drawbacks of Artificial Neural Networks are the long training times as a result of computationally expensive calculations. Sparse models aim to remove redundant parameters whilst maintaining good levels of performance. Ensembles of several weak learners have shown to be able to outperform individual networks. Crucial for the performance of the ensemble is the diversity of the individual subnetworks of which the ensemble is constructed. The work that has been done on the intersection of sparse- and ensemble learning, does not provide actual benefits in terms of computational overhead as the sparsity is simulated using binary masks. In this paper, we propose two algorithms that promote diversity among individual ensemble members. We implement these two algorithms for a first-of-its-kind Truly Sparse Ensemble [1]. We demonstrate the performance of the model at several high levels of sparsity on numerous datasets in terms of classification accuracy, Floating Point Operations (FLOPs), and running time. Moreover, we provide insight into the impact of our two diversification algorithms on the training trajectory and topological similarity of the subnetworks. Suggestions for future research are discussed as well.**

***Index Terms*—Artificial Neural Networks, Sparse Neural Networks, Ensemble Learning, Truly Sparse Networks**

I. Introduction

ARTIFICIAL Neural Networks (ANNs) have grown to be extremely popular among scholars due to an exponential increase in machine learning research over the last few decades [1]. Due to the ever-increasing amount of available data and growing complexity of problems, the demanded capability of neural networks is growing as well. Whereas expanding neural networks and thus making them more advanced will perhaps lead to better performance on the training set, networks will be more prone to overfitting and the computational overhead becomes so large that training these networks quickly becomes unfeasible. An architectural method to improve performance without risking overfitting and significantly increasing computational overhead is ensemble learning [2]. The core idea of ensemble learning is to combine several weak learners which are computationally cheap to obtain into a strong combined learner which outperforms its individual members and regularly trained single models.

Even though deeper neural networks can significantly reduce the error in training, the number of forward- and backward passes scales linearly with the number of layers resulting in undesirable long training times [3]. Moreover, for ANNs with fully-connected neurons, the number of connections increases quadratically with the number of neurons. The resources needed for training and applying these deep neural networks are thus often prohibitive [4]. Even the network size of individual learners can become so large that the time it takes to train a network quickly becomes unfeasible. In contrast, it was discovered by [5] that the more neurons a human brain has, the fewer connections between neurons are created. Transferring this concept to ANNs showed that sparse networks could obtain the same level of accuracy as their dense counterparts [6].

At the intersection of sparsity and ensemble learning, we find sparse ensembles. Despite the fact that some work has been done in this field to study the effects of pre-defined sparsity and pruning weights during training, little to no practical advantages are obtained as most sparse training is simulated by masking the weight matrices with binary masks. This method allows to study the behavior of sparse ensembles but does not decrease the computational- or memory overhead. In general, literature on sparse training has primarily been focused on the algorithmic aspect of sparsity and is based on simulating sparsity with binary masks. The reason for this focus is that almost all specialized deep learning software frameworks and hardware are optimized for dense matrix operations [7]. As a result, developing sparse deep neural networks that are not based on dense matrix operations is much more complicated than sparse networks that make use of standard (dense matrix) deep learning libraries. [8] introduced a truly sparse multilayer perceptron and showed that it was possible to train a neural network with more than one million neurons on a regular laptop without additional GPU support. Together with hardware improvements, algorithmic and software developments in the field of truly sparse training are vital to actually provide faster, energy-efficient, and memory-efficient deep neural networks.

Within this research, we aim to construct a sparse ensemble without the overhead of binary masks and want to target the lack of available literature on subnetwork diversification in sparse ensembles. Specifically, we ask ourselves the following two questions:

- *RQ1. How can we efficiently construct multilayer perceptron ensembles in the truly sparse framework?*
- *RQ2. How can we advantageously make use of dynamic sparse training properties to improve subnetwork diversity within an ensemble?*

Leveraging the insights of both [9] and [10], we introduce a first-of-its-kind truly sparse ensemble which is trained on just a single CPU core. Furthermore, we provide new additional insight into the importance of subnetwork diversity by proposing two novel algorithms

[1]Code will soon be available on: https://github.com/prdvanderwal/Diversifying-Truly-Sparse-Ensembles

that increase subnetwork diversity in ensembles. We mainly focus our efforts on tabular data as this datatype is well suited for the multilayer perceptron architecture of our ensemble's subnetworks. We summarize our contributions below:

- We introduce a first-of-its-kind Truly Sparse Multilayer Perceptron Ensemble, further demonstrating the advantages of the truly sparse framework in terms of performance, number of Floating Point Operations, and running time.
- We propose two efficient and effective subnetwork diversification algorithms for dynamic sparse ensembles that can be extended beyond the truly sparse framework.
- We make a valuable software contribution to the truly sparse framework that, for the first time, allows truly sparse networks to be stored and retrained.

Our proposed subnetwork diversification algorithms for dynamic sparse ensembles and our extension of the truly sparse framework can be considered independent contributions to the sparse neural network ensemble learning literature, and the truly sparse training literature, respectively.

II. Related Work

A. Ensemble Learning

The underlying idea of ensembles is clear; the combination of several weak learners can yield better results than a singular learner. There is a scala of ensemble methods to realize this, varying from semi-supervised- to fully supervised ensemble methods. Literature has shown that diversity among ensemble members is key for the effectiveness of ensembles [2] [11] [12] [13] [14]. The methods to establish diversity in ensembles can be categorized as either data diversity methods such as bagging [2] [15] [16] and boosting [17] [18], structural diversity methods which have different models as base predictors [2] [11], or parameter diversity methods which include evolutionary- and cross-over learning [14] [19].

Besides these three main pillars of creating high-diversity ensembles, other methods include decomposition of the data (frequently used for time series data) [11] and ensembles solely based on Negative Correlation Learning as proposed by [20].

B. Sparsity

Over the last few years, sparse implementations for several ANN architectures have been introduced among which Restricted Boltzmann Machines [21], Convolutional Neural Networks [22], and Recurrent Neural Networks [23]. An elaborate overview of applications of sparse learning in the context of deep reinforcement learning is presented in [24]. The enforced sparsity levels in literature vary greatly and can range from a moderate sparsity level of 50% as in [25], to extreme sparsity levels (>99%) as presented in [26]. Within the field of sparsity, a distinction between two main approaches to realize sparsity in an ANN can be made: pruning of the non-critical connections of the ANN during training with a dense network as a starting point [27] (dense-to-sparse training), or a sparsely initialized network which is either trained with a fixed sparse topology [21] [28] or a dynamically updating topology by pruning and regrowing weights during training [29] (sparse-to-sparse training).

Examples of dense-to-sparse training include the works of [6], [27], and [30]. However, despite a wide variety of approaches of dense-to-sparse training methods, most of these methods have in common that the computational overhead is equal to, or more than a regular dense network. An exception is the work of [31], who gradually increased the sparsity level during training whilst keeping the total memory overhead less than that of the dense implementation.

In recent years, the concept of dynamic sparse training as introduced by [29] has gained a lot of popularity among scholars. The proposed SET algorithm evolves an initially sparse topology to a sparse network reaching high levels of accuracy by pruning weights with a small magnitude and randomly regrowing the same number of weights that were just pruned. A big advantage of methods like the SET procedure [29], among which the sparse training of RNNs (ST-RNN) as presented by [32], Sparse Momentum [33], Rigged Lottery (RigL) [34], Memory-Economic Sparse training [35], and dynamic reparametrization [36], is that the network is sparsely initialized in contrast to a method like the lottery ticket hypothesis by [6] or earlier works like [27]. An alternative method was introduced by [37] centered around trainable masks. This method leaves the original weight matrix untouched when pruning during training, and by doing so preserves the historical information about parameter importance. This, however, comes at the cost of increased memory overhead.

C. Sparse Ensembles

Some work has been done at the intersection of ensemble learning and sparsity. Continuing on the work of [6], [38] used iterative magnitude pruning to create

an ensemble where subnetworks were stored at each iteration of pruning. A more recent work by [25] created ensemble members by copying a dense parent network and pruning and tuning each network individually. Yet, both of these methods apply a dense-to-sparse training regime, with significant overhead. The work of [10] presented two methods with no computational overhead during either training or testing: (Efficient) Dynamic Sparse Training Ensembles, or (E)DST Ensembles. This method entails a *global exploration* phase for a fixed number of epochs where a large part of the solution space is explored with a relatively high learning rate. After the exploration phase, the remainder of epochs is equally split into M *refinement phases*, to collect M subnetworks. During each refinement phase, the current subnetwork is refined from the previously converged subnetwork by lowering the learning rate. All new subnetworks are obtained using the Rigged Lottery (RigL) method as presented in [34]. This means that the local solution basin after convergence is escaped by pruning a large percentage of all the weights. A new subnetwork is re-initialized by regrowing the same number of weights. According to [39], 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 [6] and the SET procedure [29] on their own seem to be less effective at. Moreover, [39] showed that the lottery ticket re-initialization remains within the same basin in the solution space as the pruning solution, making the lottery tickets fundamentally limited in their ability to improve the training of sparse neural networks. This might be explained by the underlying lack of diversity in the ensemble as explained in Section II-A. A more extensive literature review on this matter can be found in Appendix A.

D. Truly Sparse Training

Little work has been done in the field of 'truly' sparse training. With truly sparse training we refer to the training of neural networks where we do not use dense matrix operations with the matrices mostly containing uselessly zero-valued weights. Almost all works in the field of sparse neural network training make use of binary masked weight matrices, which usually cause significant computational overhead.

Novel results were presented by [8] and [40] who created a truly sparse implementation of the SET algorithm for a regular multilayer perceptron and a Denoising Autoencoder respectively. The work of [8] allowed an MLP with hundreds of thousands of neurons to be trained on a regular laptop without GPU support. [9] continued this work and introduced a parallel training algorithm for truly sparse networks. To the best of our knowledge, for all existing literature making use of the truly sparse framework, it was not possible to store, re-train, and re-evaluate networks after the cache had been cleared

III. Diversifying Truly Sparse Ensembles

In this paper, we present a first-of-its-kind Truly Sparse Ensemble (TSE) at the intersection of truly sparse training and ensemble learning. At the core of this new model is our valuable software contribution that, for the first time, allows truly sparse networks to be stored and re-trained. On top of our TSE model, we propose two novel algorithms that increase diversity between subnetworks in a sparse ensemble. We implement our two algorithms in the context of a Truly Sparse Ensemble but they can be adapted for sparse ensembles based on binary masking as well.

A. Truly Sparse Ensemble

Continuing on the work of [8] and [9], we construct an ensemble of truly sparse multilayer perceptron networks. We obtain our ensemble members following the Efficient Dynamic Sparse Training (EDST) protocol as presented by [10]. We slightly diverge from the EDST implementation in [10] as a result of implementing it in a truly sparse context. Instead of regrowing weights based on the gradient magnitude when re-initializing a new subnetwork, we regrow the weights randomly as the latter would require us to calculate the gradient for all the weights in the network. During a single training run, we obtain a total of M sparse subnetworks that collectively form the TSE. For each subnetwork S_i, we define the predicted probability of the k^{th} output neuron to be $p(a_k^i)$. We get the final ensemble by stacking the output of the Softmax activation layer for all M subnetworks and averaging these probabilities, i.e., $\frac{1}{M}\sum_{i=1}^{M} p(a_k^i)$. The final prediction of the ensemble is the argumentative maximum (argmax) of the averaged probabilities of all k output neurons. We introduce several novel functionalities to the truly sparse framework to realize this, which among other things, allow networks to be stored and loaded at a later point in time. All weight matrices are stored as Compressed Sparse Row (CSR) matrices using SciPy [41].

B. Subnetwork Diversification

The importance of subnetwork diversity for the performance of an overall ensemble has repetitively been

emphasized in literature [2] [11] [12] [13] [14]. We propose two new algorithms focused on improving subnetwork diversity in an ensemble: Distance EDST and Disjoint EDST. The novelty of both methods lies in how we grow new connections when re-initializing a new subnetwork. In contrast to the original EDST procedure, for Distance EDST and Disjoint EDST, a subnetwork's weights are grown back based on the euclidian distance to its predecessor or the topological dissimilarity to all predecessors, respectively. Moreover, we introduce a new implementation of the EDST refinement phase, focusing on more than just learning rate reduction.

1) Distance Re-initialization: In contrast to the original EDST algorithm, which does not actively try to promote diversification among subnetworks when regrowing weights, Distance-EDST (D-EDST) forces the network to regrow the weights in such a way that the euclidean distance between the weights of the new subnetwork and the existing subnetwork, is at least a certain distance within the solution space. 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. See Appendix G for more details. Given this property, we use a sampled mean and a factored standard deviation as indicators to find sufficiently distant weight matrices. An overview of D-EDST can be found in Algorithm 1.

Our proposed algorithm iteratively finds a sufficiently distant weight matrix for all the hidden layers in the multilayer perceptron subnetwork. The minimum euclidean distance threshold for a generated weight matrix to be accepted as a weight matrix of the new subnetwork is established by generating 100 random weight matrices and calculating the mean and standard deviation of the euclidean distance. We calculate the euclidean distance using Scikit Learn's [42] pairwise euclidean distance method as it is efficient in dealing with sparse data. The distance between each row vector j in the converged subnetwork's weight matrix (W_i) and the *proposedWeights* matrix (P_i) is calculated as follows:

$$Distance(W_i^j, P_i^j) = \sqrt{((W_i^j \cdot W_i^j) - 2 \times (W_i^j \cdot P_i^j) + (P_i^j \cdot P_i^j))} \quad (1)$$

where i is the weight matrix index (layer of the network), and j the row index. The final euclidean distance is the total sum of the matrix resulting from Equation 1.

We select the value of 100 as this was the lowest value for which the distribution of euclidean distances sufficiently followed a Gaussian distribution (see Appendix G), making the mean and standard deviation suitable indicators. 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 is presented in Figure 1. We observe that the running time strongly increases for values of the matrix distance coefficient which are larger than 2.0.

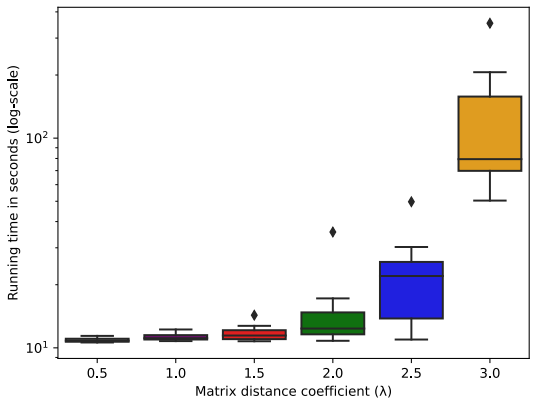


Fig. 1: Boxplot of the impact of the matrix distance coefficient on the running time for finding a suitably distant matrix. For each value of λ, we ran 10 iterations of the Distance Re-Initialization algorithm on the Gesture Phase Segmentation dataset (see Section IV-A) for a 1000x1000 weight matrix at a sparsity level of 0.96.

Algorithm 1: Truly Sparse EDST Ensemble with Distance Re-Initialization

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

for $i \leftarrow 2$ **to** $k-1$ **do**
  $\boldsymbol{W}_i' \leftarrow$ 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 $\leftarrow 0$
  iteration $\leftarrow 0$
  while *currentDistance* $< (\mu + \lambda \times \sigma)$ **do**
    if *iteration* $>$ *maxIterations* **then**
      Throw Iteration Error and stop training
    proposedWeights $\leftarrow \boldsymbol{W}_i' \cup$ RegrowRandomK(q)
    currentDistance $\leftarrow$ CalculateDistance($\boldsymbol{W}_i$, proposedWeights)
    iteration += 1
  end
  $\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i' \cup$ proposedWeights
end

Algorithm 2: Truly Sparse EDST Ensemble with Disjoint Re-Initialization

Data: Layer i to k with: Sparse Weight Matrix $\mathbf{W}_i$ and global exploration rate q

1 **for** $i \leftarrow 2$ **to** $k-1$ **do**
2 **for** $j \leftarrow 0$ **to** $\boldsymbol{W}_i$.numRows() **do**
3 **Blocked**$_i$.add(tuple($\boldsymbol{W}_i$.row[j] , $\boldsymbol{W}_i$.col[j]))
4 **end**
5 $\boldsymbol{W}_i' \leftarrow$ PruneSmallestK($|\boldsymbol{W}_i|$, q)
6 numWeightsRegrown $\leftarrow 0$
7 iteration $\leftarrow 0$
8 **while** numWeightsRegrown $\neq K$ **do**
9 proposedWeights $\leftarrow$ (RegrowRandomK(q) - numWeightsRegrown)
10 **for** $j \leftarrow 0$ **to** $\boldsymbol{W}_i$.numRows() **do**
11 **if** *iteration* $>$ *maxIterations* **then**
12 store(proposedWeights.numRows())
13 **break**
14 **else if** tuple(proposedWeights.row[j], proposedWeights.col[j]) in **Blocked**$_i$ **then**
15 proposedWeights.delete(j)
16 **end**
17 $\boldsymbol{W}_i' \leftarrow \boldsymbol{W}_i' \cup$ proposedWeights
18 numWeightsRegrown += proposedWeights.numRows()
19 iteration += 1
20 **end**
21 $\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i' \cup$ proposedWeights
22 **end**

Taking into consideration the objective of keeping additional computational overhead and running time to a minimum, we have limited the experiments for the D-EDST implementation to 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. The integration of this algorithm in the original EDST procedure can be found in Appendix 3.

2) Disjoint Re-initialization: The Disjoint EDST algorithm focuses on improving topological diversity among subnetworks when re-growing weights for the re-initialization of a new subnetwork. Instead of doing this without care for diversity like the original EDST algorithm, or by finding a more distant weight matrix like we do for D-EDST, we iteratively regrow weights that are not part of the final topology of any previous subnetwork. We do this by saving all the row- and column indices of the converged final state of the subnetworks and storing these per layer in a set (datatype). When the new model is initialized, weights are randomly regrown as long as they are not in the set of "blocked" weights of the previous subnetwork(s). Given that the time complexity of a look-up in a set is *O(1)*, the computational overhead of this iterative process is marginal. An overview of the Disjoint EDST implementation can be found in Algorithm 2. Important to note is that we make use of SciPy's [41] sparse matrix COOrdinate (COO) format for all weight matrices in this algorithm. This format stores the row- and column indices for each individual value. This means that the $.row[]$ and $.col[]$ calls in lines 3 and 14 of Algorithm 2 return one-dimensional vectors of identical length over which we iterate.

The proposed disjoint EDST algorithm has two variations: (1) Regular Disjoint EDST (RD-EDST) and (2) Fully Disjoint EDST (FD-EDST). For the first variation, 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 also propose the second variation where the regrown weights in the new subnetwork are fully disjoint with any previous subnetwork. A schematic overview of FD-EDST can be found in Figure 2. Forcing the networks to not regrow weights in the topological locations similar to the final state of previous networks, is done by the same iterative look-up process within the set of blocked weights for the regular weight evolution procedure (part of the SET training regime) as well. Thus, the only difference between RD-EDST and FD-EDST is step c of Figure 2. We only apply the disjoint re-initialization to the weight matrices of the hidden layers. The reason for this is that all weight matrices are initialized following the Erdős-Rényi distribution as presented in [29]. As a result, for datasets with a relatively low number of input features and output classes, the first- and last layer of a subnetwork are too dense to find disjoint matrices. For consistency, we also apply the D-EDST algorithm only to the weight matrices of the hidden layers.

3) Refinement Phase: The original EDST refinement phase as presented in [10] only lowers the learning rate to allow the subnetwork to converge to a better solution. Extending the idea of lowering the learning rate to reach better convergence, we propose to also lower both the pruning rate and the frequency with which the weight evolution procedure is applied (evolution frequency). The aim of our proposed refinement phase, the *Comprehensive Refinement Phase*, is that the network is able to converge better during the refinement phase as it trains for more epochs with the same topology and is disrupted less as a smaller percentage of the weights is pruned and regrown.

More specifically, after the exploration phase, each subnetwork will have a two-stepped refinement phase. In the first step, only the frequency with which we apply the weight evolution procedure is halved. In the second step of the refinement phase, we half the learning rate, pruning rate, and evolution frequency once more. After a pre-determined amount of epochs, we obtain the final topology of the subnetwork and enter the cycle of

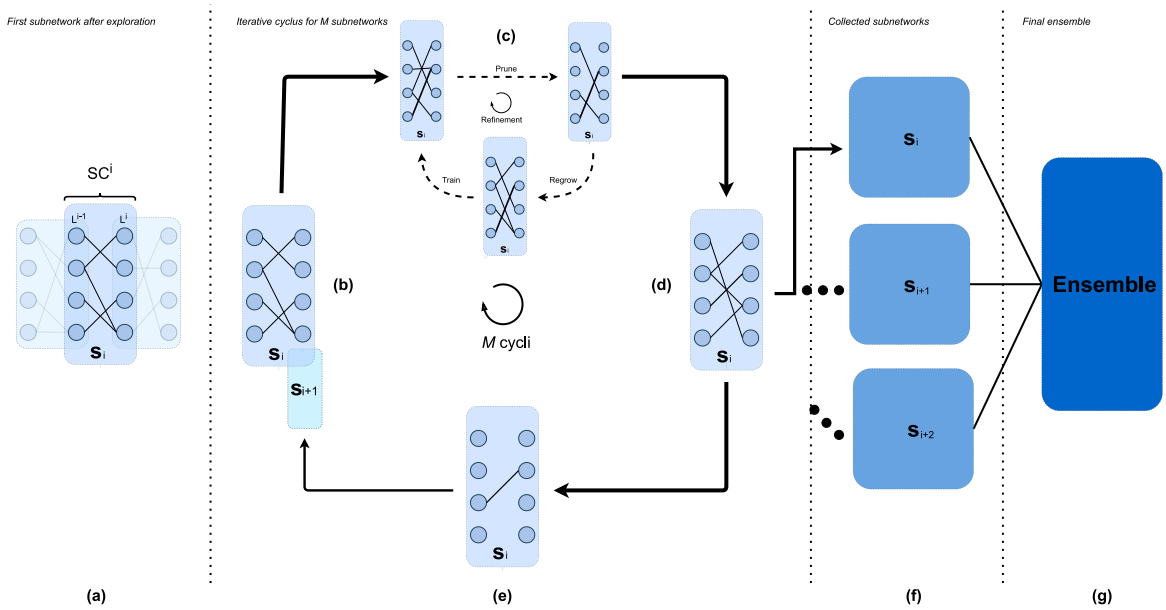


Fig. 2: Schematic overview of the Fully Disjoint EDST (FD-EDST) training procedure. The procedure is visualized for a single sparse connected layer (SC^i) but is applied to all hidden layers. After the initial exploration phase, we obtain the base subnetwork S_i (a). We take this network (b) and start 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 (d) which we store for later (f). In order 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 will now be repeated another $M-1$ times to collect M almost fully disjoint subnetworks which we combine into an ensemble (g).

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.

IV. Experimental evaluation

In this section, we demonstrate the successful implementation of the Truly Sparse Ensemble in terms of accuracy, number of parameters, training Floating Point Operations (FLOPs), and training time. We compare our two proposed algorithms with the existing EDST implementation and several baselines. We evaluate the effectiveness of our algorithms in terms of diversity by visualizing the training trajectory and topological distance.

A. Datasets

We train and evaluate all models on three publicly available datasets and the tabular benchmark established by [43], all from different domains. A summary of the dataset characteristics for the three selected datasets can be found in Table I. For all ensemble experiments, we use 20% of the training dataset as a validation set to evaluate the subnetworks during training and determine when to save a subnetwork. The original Higgs dataset has 11000000 samples but we have decided to take a subset and select the same number of samples as [9]. Even though the focus of this research is on tabular data, we include one image dataset which makes it slightly easier to understand what is happening. We normalized all datasets.

1) CIFAR-10: The CIFAR-10 dataset [44] is a widely used benchmark dataset, mostly in the field of computer vision. The dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. The

TABLE I: Summary of datasets used.

Dataset	Dataset properties					
	Domain	**Features**	**Train samples**	**Test samples**	**Classes**	**Weight initialization**
CIFAR-10	RGB Images	3072	50000	10000	10	He Uniform
Gesture Phase Segmentation	Video Segmentation	50	7898	1975	5	He Uniform
HIGGS	Physics particles	28	105000	50000	2	Xavier

classes entail various types of vehicles and animals. Given that all experiments in this paper are based on MLPs, we flatten the RGB channels, ending up with an input size of 3072.

2) Gesture Phase Segmentation: The Gesture Phase Segmentation dataset (GPS), obtained from the UCI Machine Learning Repository [45], includes both raw and processed data from 7 videos with people gesticulating, with the aim of studying gesture phase segmentation. After pre-processing (scaling and removing missing values), we end up with 9873 instances and 50 features.

3) HIGGS: We obtained the HIGGS dataset from the UCI Machine Learning Repository just like GPS. The dataset consists of 11000000 samples which were created using Monte Carlo simulations. All samples belong to either of the two classes: particle or no particle. We make use of all 28 features, of which 21 are low-level features (kinematic measurements) and 7 are high-level features that were manually derived by physicists. As mentioned above, we only use 105000 training samples and 50000 testing samples due to constraints in computational capacity.

Even though MLPs can also be used for data types like images, a lot of spatial information is lost when the images are flattened. A convolutional neural network is generally speaking much better suited to capture such spatial relations. Tabular data, on the other hand, should theoretically be an ideal data type for MLPs as no information is lost and the data can directly be used as input for the model. Responding to the strong lack of benchmark tabular datasets (e.g. MNIST and CIFAR for image recognition), [43] provide a first-of-its-kind tabular benchmark, consisting of 45 datasets [2]. We evaluate how well our models perform on a collection of datasets without fine-tuning our model's hyperparameters. We test our proposed methods and baselines on 14 numerical classification tabular datasets, as selected by [43].

[2]Repository last visited on 28-02-2023. Datasets might be updated/changed. Link: https://huggingface.co/datasets/inria-soda/tabular-benchmark

B. 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 training time in minutes. In contrast to most related work, we do not calculate the number of training FLOPs theoretically for our truly sparse models. Instead, we make use of the Performance Application Programming Interface software [46]. This library provides us with the interface and methodology for a hardware performance counter, allowing us to access and process raw CPU data and extract the actual number of FLOPs performed by the CPU.

C. Experimental Setup

We constructed all models with three hidden layers, each consisting of 1000 neurons. For all models, we applied the Alternated Left ReLU (Alt-ReLU) activation function [9] to all layers except for the last layer, for which we used the Softmax activation function. Moreover, all models were trained for a total of 350 epochs with a batch size of 128. All ensembles, unless indicated differently, consist of five submodels. The training procedure entailed an exploration phase of 100 epochs with a learning rate of 0.1, followed by a two-stepped refinement phase with a learning rate of 0.1 and 0.05, respectively. The values for these hyperparameters were selected using a small grid search. As discussed in Section III-B3, we also halve the frequency with which we apply an adaptive training step and the number of weights we prune and regrow during these steps. We use Stochastic Gradient Descent (SGD) with a momentum of 0.9 as the optimization method. The initial evolution frequency with which we perform a topology update is every two epochs. The selected weight initialization method differed per dataset and can be found in Table I. All layers in each network were initialized with the Erdős-Rényi (ER) distribution at a sparsity level of S as presented in [29]. The hyperparameter ϵ determines the sparsity level of a model layer. The ER distribution allocates higher sparsity to the layers with more neurons. All our sparse models were put under very high sparsity

constraints of 0.93 and up. For most datasets, we used varying values of ϵ for our experiments. However, as a result of the aforementioned ER initialization property, the sparsity level slightly differs per datatset. For our dense models, we were not able to run experiments with a fixed learning rate of 0.01 as presented in [29] due to the numerical instability of the truly sparse framework. For HIGGS and the Gesture Phase Segmentation dataset, we used a fixed learning rate of 0.001 and for CIFAR-10 we were forced to use an even smaller learning rate, namely 0.0001. Moreover, we used the regular ReLU activation function for all dense experiments for the same reason. In all tables, the overall accuracy of the network is reported. To allow for reproducibility of the results, all experiments were run on a single core of a Dell T60 (remote) server with a 2xSilver-4210 processor. An overview of all hyperparameter configurations can be found in Appendix E.

D. Baselines

We consider EDST [10] to be the closest baseline to our proposed models. However, the original EDST model is implemented in PyTorch [47], a well-established and highly optimized machine learning framework. Given that a comparison of our models in terms of FLOPs and running time to a model in PyTorch yields a distorted impression of the effectiveness of our algorithms, we instead implement the original EDST in the truly sparse framework with the minor adjustment as discussed in Section III-A. We also compare our proposed methods with a dense MLP, a dense ensemble, a single SET-MLP [29], and a single static sparse model. We implement all of these models in the truly sparse framework to allow for an accurate and fair comparison.

E. Experimental Results

We demonstrate the performance of our proposed EDST Truly Sparse Ensembles (TSE) by comparing it to the baseline models described above. We evaluate the following models: the original EDST TSE (EDST), Regular Disjoint EDST TSE (RD-EDST), Fully Disjoint EDST TSE (FD-EDST), and Distance EDST TSE (D-EDST). An overview of a selection of the experimental results on the CIFAR-10-, Gesture Phase Segmentation-, and HIGGS dataset can be found in Tables II, III, and IV. An overview of all experimental results on these datasets can be found in Appendix D.

Analyzing Tables II, III, and IV, we distinguish between the results on the CIFAR-10 dataset and the Gesture Phase Segmentation- and Higgs datasets due to the nature of their data types. We observe that for the CIFAR-10 dataset the original EDST implementation slightly outperforms our proposed methods. Moreover, it becomes clear that almost all models at a higher sparsity level (e.g. 0.98) outperform those trained on a lower sparsity level. Even though sparse models have been shown to outperform dense models [10], at these extremely high levels of sparsity, it is expected that having some more weights would benefit the network's performance. We observe a similar pattern in Table IV.

TABLE II: Summary of experiments of our EDST implementations and baselines on the CIFAR-10 dataset. 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.

Architecture	**Model**	**Sparsity**	**Results**			
			Accuracy [%]	**Weights [#]**	**Train Flops [#]**	**Train time [min]**
3072-1000-1000-1000-10	Single Dense Model	-	57.0	5,085,010	2.18e14	$\sim$ 693.1
	EDST	0.98	**62.8**	0.08x	0.03x	$\sim$ 138.3
		0.97	62.0	0.16x	0.05x	$\sim$ 178.3
		0.95	61.8	0.24x	0.08x	$\sim$ 290.9
	RD-EDST (ours)	0.98	61.6	0.08x	0.03x	$\sim$ 119.7
		0.97	61.5	0.16x	0.05x	$\sim$ 230.1
		0.95	61.6	0.24x	0.08x	$\sim$ 275.7
	FD-EDST (ours)	0.98	61.3	0.08x	0.03x	$\sim$ 135.2
		0.97	60.7	0.16x	0.05x	$\sim$ 231.3
		0.95	60.6	0.24x	0.08x	$\sim$ 312.9
	D-EDST ($\lambda = 1.5$) (ours)	0.98	62.6	0.08x	0.03x	$\sim$ 121.5
		0.97	61.0	0.16x	0.05x	$\sim$ 225.0
		0.95	61.9	0.24x	0.08x	$\sim$ 282.3
	SET-MLP	0.97	59.2	0.03x	0.05x	$\sim$ 216.4
	Single Static Sparse Model	0.97	57.1	0.03x	0.05x	$\sim$ 206.4
	Dense Ensemble	-	61.0	5.00x	5.00x	$\sim$ 3234.2

TABLE III: Summary of experiments of our EDST implementations and baselines on the Gesture Phase Segmentation dataset.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.

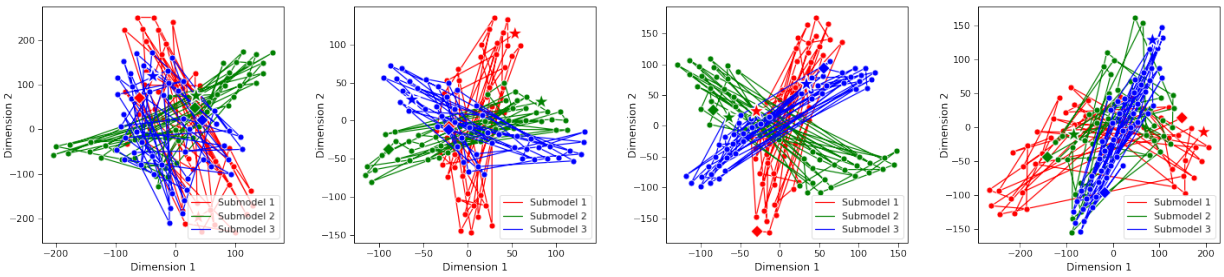
Architecture	**Model**	**Sparsity**	**Results**			
			Accuracy [%]	**Weights [#]**	**Train Flops [#]**	**Train time [min]**
50-1000-1000-1000-5	Single Dense Model	-	54.9	2,058,005	1.71e13	$\sim$ 60.0
	EDST	0.98	71.8	0.12x	0.04x	$\sim$ 16.8
		0.95	72.2	0.25x	0.07x	$\sim$ 24.6
		0.93	72.5	0.37x	0.11x	$\sim$ 33.0
	RD-EDST (ours)	0.98	71.8	0.12x	0.04x	$\sim$ 17.1
		0.95	72.6	0.25x	0.07x	$\sim$ 31.1
		0.93	72.9	0.37x	0.11x	$\sim$ 53.1
	FD-EDST (ours)	0.98	71.6	0.12x	0.04x	$\sim$ 18.1
		0.95	74.2	0.25x	0.07x	$\sim$ 36.9
		0.93	**75.3**	0.37x	0.11x	$\sim$ 74.0
	D-EDST ($\lambda = 1.5$) (ours)	0.98	72.0	0.12x	0.04x	$\sim$ 16.2
		0.95	73.4	0.25x	0.07x	$\sim$ 25.7
		0.93	73.5	0.37x	0.11x	$\sim$ 32.2
	SET-MLP	0.95	60.4	0.05x	0.07x	$\sim$ 25.6
	Single Static Sparse Model	0.95	67.4	0.05x	0.07x	$\sim$ 22.7
	Dense Ensemble	-	54.0	5.00x	5.00x	$\sim$ 299.9

TABLE IV: Summary of experiments of our EDST implementations and baselines on the HIGGS dataset. 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.

Architecture	**Model**	**Sparsity**	**Results**			
			Accuracy [%]	**Weights [#]**	**Train Flops [#]**	**Train time [min]**
28-1000-1000-1000-2	Single Dense Model	-	57.1	2,033,002	2.28e14	$\sim$ 709.6
	EDST	0.98	53.1	0.12x	0.04x	$\sim$ 241.6
		0.95	53.1	0.25x	0.07x	$\sim$ 330.1
	RD-EDST (ours)	0.98	64.3	0.12x	0.04x	$\sim$ 222.7
		0.95	63.5	0.25x	0.07x	$\sim$ 316.5
	FD-EDST (ours)	0.98	64.0	0.12x	0.04x	$\sim$ 229.1
		0.95	63.7	0.25x	0.07x	$\sim$ 339.2
	D-EDST ($\lambda = 1.5$) (ours)	0.98	**64.5**	0.12x	0.04x	$\sim$ 232.0
		0.95	63.5	0.25x	0.07x	$\sim$ 343.0
	SET-MLP	0.95	54.3	0.05x	0.07x	$\sim$ 332.1
	Single Static Sparse Model	0.95	53.1	0.05x	0.07x	$\sim$ 315.4
	Dense Ensemble	-	60.4	5.00x	5.00x	$\sim$ 3561.5

We consider an in-depth analysis of the sparsity-accuracy trade-off in sparse ensembles to be out of the scope of this research, for which reason we leave this to future work.

For the two tabular datasets we observe that for almost all variations, the two proposed algorithms consistently outperform the original EDST implementation and all other baselines without any significant additional overhead in terms of training FLOPS and training time. Interestingly, the best-performing method is different for either 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 proposed 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 (see Appendix G). FD-EDST was the only model that benefited from an increase in the number of subnetworks. The results in Tables III, and IV show us that the proposed algorithms and variations improve overall ensemble performance on the two selected tabular datasets. Yet, further research is required to get a better understanding of the underlying relation between the dataset characteristics and the individual proposed methods. Experiments at lower sparsity levels were not possible due to the numerical instability of the truly sparse framework.



(a) EDST (b) Regular Disjoint EDST (c) Fully Disjoint EDST (d) Distance EDST ($\lambda = 1.5$)

Fig. 3: t-SNE projection of the training trajectories of three subnetworks discovered by various EDST implementations on the Gesture Phase Segmentation dataset. The diamonds and stars represent the start and end of each subnetwork respectively. The sparsity level is $S = 0.95$.

TABLE V: Aggregated results of experiments on 14 numerical classification datasets from the tabular data benchmark [43]. The Best Performing Model (BPM) metric represents the number of times that a model got the highest classification accuracy of all models.

Dataset	**Epsilon (ϵ)**	**Model**	**BPM [#]**	**Total training time [min]**
Aggregated	-	Single Dense Model	0	$\sim$ 2011
datasets	20	EDST	2	$\sim$ 827
		RD-EDST (ours)	2.33	$\sim$ 899
		FD-EDST (ours)	2.33	$\sim$ 1010
		D-EDST ($\lambda = 1.5$) (ours)	**6.33**	$\sim$ 823
		SET-MLP	1	$\sim$ 820
		Single Static Sparse Model	0	$\sim$ 789
	-	Dense Ensemble	-	-

F. Aggregated Results

In order to test how well our proposed methods perform without dataset-specific fine-tuning, we evaluate all models on the numerical classification benchmark for tabular data [43]. For the training of all models, we used the same configuration as described in Section IV-C. 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. In Table V, the frequency of the best-performing model and total training time on the numerical classification benchmark for tabular data [43] is presented. For more details on the individual datasets, please refer to Appendix H. From Table V, we observe that for 11 out of 14 datasets, one of our proposed methods achieves the highest classification accuracy, with D-EDST reaching the highest classification accuracy most frequently. Both disjoint implementations have a slight overhead in terms of running time but remain far below the running time of a single dense model. Evaluating a dense ensemble on all 14 datasets proved to be infeasible due to the time it would take to do a single experiment. Extrapolating the observed multiplicative factor for the running time in minutes from Tables II, III, and IV, it would take nearly 170 hours for a single run for a Dense Ensemble. The results in Table V confirm that our proposed diversification algorithms positively impact overall ensemble performance even when applied to a wide range of tabular datasets.

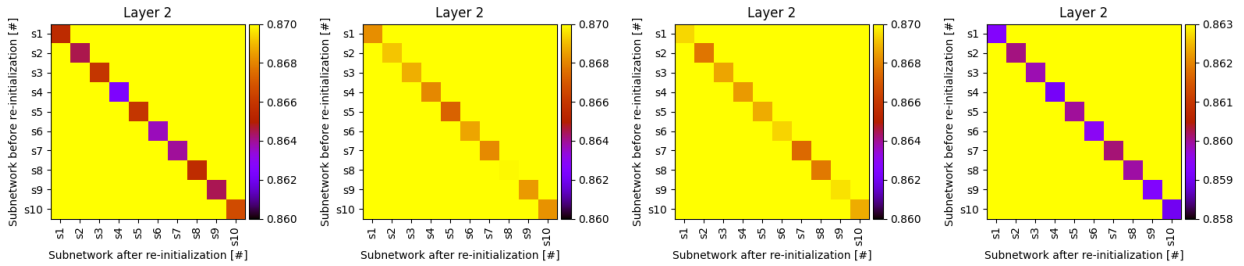
G. Training Trajectory

In order to get a better understanding of the effectiveness of the two proposed algorithms and their variations (RD-EDST, FD-EDST, and D-EDST), we use t-SNE [48] to visualize the training trajectories of our submodels in the solution space. We do this by saving the Softmax output on the test data for each epoch and reducing the dimensionality to 2D using t-SNE. Figure 3 gives an overview of the different training trajectories of the three subnetworks on the Gesture Phase Segmentation dataset. We observe that the trajectories of the subnetworks of the original EDST implementation (3a) seem fairly random

and often have shared trajectories. For Figure 3b, we see that the training trajectory of a new subnetwork is dissimilar to the previous networks and that the model is somewhat able to maintain this difference in direction. For Figure 3c, we observe that throughout every refinement phase for each subnetwork, the trajectories remain separated, indicating the effectiveness of the algorithm to obtain disjoint networks. The random and shared trajectories visible in Figure 3d were to be expected as there is no regularization in place for the model to take a different trajectory direction besides random re-initialization (in terms of topological similarity). It is important to note that the distance between points does not tell us much per se. A lot of information might be lost in the dimensionality reduction as t-SNE is not guaranteed to preserve global distances.

H. Topological Distance between Subnetworks

Building on the findings of [49], we visualize the topological distance between subnetworks for each proposed method. 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 original EDST 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 Figure 4, a visualization of the topological distances between the converged subnetworks and re-initialized subnetworks is presented. From the values on all the color scales it becomes immediately clear that the topologies of all the models are very dissimilar given that all scores are in the upper region of 0.8 (a value of 1 represents complete dissimilarity). This is however to be expected as we are dealing with extremely sparse networks. It is thus very likely that the topologies differ greatly to start with and that the margins of topology difference with varying methods are thus relatively small. Nevertheless, we observe from Figures 4b and 4c 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 original EDST (figure 4a). We also see that the Distance EDST algorithm (Figure 4d) consistently yields topologies that are closer to each other compared to the original implementation. Given that the Distance EDST implementation focuses on increasing the euclidean distance between subnetworks in the solution space (weight values) and not necessarily the distance between topologies, it is not strange that the topologies are not more distant than the original EDST implementation. Yet, the phenomenon that D-EDST actually yields subnetworks that are topologically closer to their predecessors compared to the original EDST implementation is surprising. 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 its predecessor had a positive weight and the other way around. As the



(a) EDST (b) Regular Disjoint EDST (c) Fully Disjoint EDST (d) Distance EDST ($\lambda = 1.5$)

Fig. 4: Heatmap representing the topological distance (NNSTD [49]) 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 EDST implementations on the Gesture Phase Segmentation dataset. $w_i (i = 0, 1, ..., 9)$ represent 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, whilst a value of 1 means that the models' topologies are completely different. The sparsity level is $S = 0.95$.

TABLE VI: Ablation study on the impact of the Comprehensive Refinement phase on CIFAR-10, the Gesture Phase Segementation (GPS) dataset, and HIGGS dataset.

Model	Comprehensive Refinement	Results		
		CIFAR-10 Acc. [%]	GPS Acc. [%]	HIGGS Acc. [%]
RD-EDST (ours)	No	**62.1**	71.2	**63.7**
	Yes	61.5	**72.6**	63.5
FD-EDST (ours)	No	**61.0**	73.9	63.5
	Yes	60.7	**74.2**	**63.7**
D-EDST ($\lambda = 1.5$) (ours)	No	**62.3**	71.7	63.5
	Yes	61.0	**73.4**	**63.5**

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.

We further provide visualizations of the topological distance between converged subnetworks that were trained on CIFAR-10, the Gesture Phase Segmentation dataset, and the HIGGS dataset in Appendix I. For the Gesture Phase Segmentation dataset we observe that for our proposed methods the topologies of the converged subnetworks are less distant than for the original EDST implementation. This might also be an explanation for the relatively small positive difference in classification accuracy between our proposed methods and the original EDST implementation on the Gesture Phase Segmentation Dataset. We want to emphasize that the goal of our diversification methods is not to just create subnetwork diversity in terms of topology, but also diversity in functionality (i.e. what a subnetwork actually predicts). Moreover, we use the relatively new NNSTD [49] metric to measure the topological distance between our subnetworks which might still have some undiscovered limitations. Also, this metric does not factor in other diversity characteristics such as the values of the weights, and is thus not an absolute indicator of diversity.

I. Ablation Study Refinement Phase

Next to our two proposed diversification algorithms, we also suggest reducing the number of weights we prune and regrow during the SET procedure and 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 in [10] where only the learning rate is halved during the two-stepped refinement phase, to our proposed *Comprehensive Refinement Phase*. As elaborated on in Section III-B3, we only halve the evolution frequency during the first part of the refinement phase, and halve the learning rate, the fraction of weights we prune and regrow, and the evolution frequency once more, in the second part of our refinement phase. The results of the ablation study can be found in Table VI.

From Table VI we observe that for the tabular datasets, our proposed refinement phase is predominantly beneficial. Similar to the results presented in Section IV-E, the results on CIFAR-10 deviate from those on the tabular datasets. For CIFAR-10, all models achieved better performance with just the learning rate reduction. We did not observe any significant difference in terms of FLOPs and running time for the ablation experiments (see Appendix J). Further experiments are required to get a better understanding of the difference in behavior when our models are trained and evaluated on an image dataset instead of tabular data. We leave this to future work.

V. Conclusion and Future Work

This research proposes two novel algorithmic methods to improve subnetwork diversity for ensembles in a truly sparse context. At the intersection of truly sparse training and ensemble learning, we successfully create a first-of-its-kind truly sparse multilayer perceptron ensemble. To realize this, we make a valuable software contribution that, for the first time, allows truly sparse networks to be stored and re-trained. With a main focus on tabular data, the learning capabilities of the truly sparse ensemble were evaluated on numerous tabular datasets and the CIFAR-10 image dataset for improved intuition. Our two proposed algorithms introduce a new perspective on subnetwork diversification in dynamic sparse ensembles where the novelty of both methods lies in how we grow new connections when re-initializing a new subnetwork. The effectiveness of our diversification methods is assessed in terms of classification accuracy and additional

insights are derived by visualizing the topological similarity and training trajectory of the subnetworks.

Our diversification algorithms are centered around increased parameter- and topological diversity when re-initializing a new subnetwork. We increased parameter diversity by re-initializing subnetworks in such a way that the euclidian distance between the weights of the new subnetwork and the existing subnetwork is more than the minimum threshold distance in the solution space. Alternatively, we introduced a novel constraint for our dynamic sparse training procedure where newly initialized subnetworks are- and remain disjoint with all preceding subnetworks. We found that for both the Gesture Phase Segmentation- and HIGGS dataset, and for the newly introduced tabular data benchmark [43], our proposed methods outperform practically all baseline models without almost any computational overhead in terms of Floating Point Operations and running time. All models in this paper were trained and evaluated on a single CPU core.

Limitations. All experiments presented in this paper were model implementations in the truly sparse network. Due to the numerical instability of the network, the range and variety of experiment configurations were quite limited. As a result, we were, for example, not able to test the effectiveness of our diversification methods at lower levels of sparsity. Moreover, we implemented our baseline models in the truly sparse framework to allow for a better comparison with our proposed models. This, however, makes comparing our results to other non-truly sparse training literature difficult. Also, the number of metrics on which we evaluated our models were limited and additional metrics could have provided additional insights and a deeper understanding.

The experiments presented in this paper have provided novel insights into sparse ensemble diversification and contributed to the further development of the truly sparse framework. Nevertheless, more experiments should be conducted to get a better understanding of our diversification algorithms in varying settings such as lower sparsity levels and other datasets. It would also be interesting to evaluate the effectiveness of our proposed algorithms in the setting of sparse models that are based on binary masking. The numerical instability of the truly sparse framework should be tackled as it is currently a limiting factor for model development and experimentation with the truly sparse framework.

Our proposed algorithms regrow weights during re-initialization with the mere focus to increase subnetwork diversity. The recent works of [10], [34], and [39], have demonstrated the high effectiveness of regrowing weights based on gradient magnitude. This poses a challenge for truly sparse ensemble learning. In order to regrow weights based on gradient magnitude, the gradients for a dense matrix need to be computed, defying the benefit of a truly sparse network. Given the success of gradient-based re-growing of weights in sparse ensembles, future research should investigate alternatives for truly sparse training. Other training configurations that are worth exploring include the use of different heuristics for the reduction of hyperparameters like the evolution frequency and learning rate, during the refinement phase. A possible alternative could be to use cosine annealing as presented by [34].

Quantifying subnetwork diversity remains complicated as it is a multifaceted problem, and includes, but is not limited to, topology- and parameter diversity. Future efforts could be focused on the development of a new, more comprehensive diversity metric that could encompass the aforementioned two metrics, but also other metrics like the Kullback-Leibler Divergence- [50] and Prediction Disagreement between subnetworks as presented in [10].

To conclude, within this work, two novel effective ensemble diversification methods have been introduced and a state-of-the-art truly sparse ensemble was successfully designed and implemented by extension of the truly sparse framework. Yet, the success of the truly sparse framework and potential future introduction of actual faster, and memory-efficient neural networks, will greatly depend on the further development of designated sparse hardware platforms and software libraries.

ACKNOWLEDGEMENT

I am extremely grateful to dr. Decebal Mocanu for his guidance, positive attitude, and extensive feedback which helped me greatly to determine the course of this research and overcome several challenges along the way. Our weekly meetings have kept me motivated and inspired throughout the entire process. I also wish to thank dr. Nicola Strisciuglio and dr. Yanqui Huang for their refreshing perspective and valuable input.

REFERENCES

[1] T. G. Dietterich, "Machine-learning research," *AI magazine*, vol. 18, no. 4, pp. 97–97, 1997.

[2] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, no. 2, pp. 241–258, Apr. 2020. [Online]. Available: http://link.springer.com/10.1007/s11704-019-8208-z

[3] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth. arxiv 2016," *arXiv preprint arXiv:1603.09382*.

[4] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *CoRR*, vol. abs/1902.09574, 2019. [Online]. Available: http://arxiv.org/abs/1902.09574

[5] S. Herculano-Houzel, B. Mota, P. Wong, and J. H. Kaas, "Connectivity-driven white matter scaling and folding in primate cerebral cortex," *Proceedings of the National Academy of Sciences*, vol. 107, no. 44, pp. 19 008–19 013, 2010.

[6] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," Feb. 2018. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[7] G. Sokar, Z. Atashgahi, M. Pechenizkiy, and D. C. Mocanu, "Where to Pay Attention in Sparse Training for Feature Selection?" Oct. 2022. [Online]. Available: https://openreview.net/forum?id=xWvI9z37Xd

[8] S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, and M. Pechenizkiy, "Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware," *Neural Computing and Applications*, vol. 33, no. 7, pp. 2589–2604, Apr. 2021. [Online]. Available: https://link.springer.com/10.1007/s00521-020-05136-7

[9] S. Curci, D. C. Mocanu, and M. Pechenizkiyi, "Truly sparse neural networks at scale," *arXiv preprint arXiv:2102.01732*, 2021.

[10] S. Liu, T. Chen, Z. Atashgahi, X. Chen, G. Sokar, E. Mocanu, M. Pechenizkiy, Z. Wang, and D. C. Mocanu, "Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity," in *ICLR*, 2022.

[11] Y. Ren, L. Zhang, and P. Suganthan, "Ensemble Classification and Regression-Recent Developments, Applications and Future Directions [Review Article]," *IEEE Computational Intelligence Magazine*, vol. 11, no. 1, pp. 41–53, Feb. 2016, conference Name: IEEE Computational Intelligence Magazine.

[12] O. Sagi and L. Rokach, "Ensemble learning: A survey," *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1249. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1249

[13] A. Chandra and X. Yao, "DIVACE: Diverse and Accurate Ensemble Learning Algorithm," in *Intelligent Data Engineering and Automated Learning – IDEAL 2004*, ser. Lecture Notes in Computer Science, Z. R. Yang, H. Yin, and R. M. Everson, Eds. Berlin, Heidelberg: Springer, 2004, pp. 619–625.

[14] S. Chen, R. Zhao, and H. Fu, "Ensemble diversity enhancement based on parameters evolution of base learners," in *2021 33rd Chinese Control and Decision Conference (CCDC)*, May 2021, pp. 5887–5893, iSSN: 1948-9447.

[15] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996. [Online]. Available: https://doi.org/10.1007/BF00058655

[16] G. Webb and Z. Zheng, "Multistrategy Ensemble Learning: Reducing Error by Combining Ensemble Learning Techniques," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, pp. 980–991, Sep. 2004.

[17] Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm," *undefined*, 1996. [Online]. Available: https://www.semanticscholar.org/paper/Experiments-with-a-New-Boosting-Algorithm-Freund-Schapire/68c1bfe375dde46777fe1ac8f3636fb651e3f0f8

[18] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, Aug. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002200009791504X

[19] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why m heads are better than one: Training a diverse ensemble of deep networks," *arXiv preprint arXiv:1511.06314*, 2015.

[20] Y. Liu and X. Yao, "Ensemble learning via negative correlation," *Neural Networks*, vol. 12, no. 10, pp. 1399–1404, Dec. 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608099000738

[21] D. C. Mocanu, E. Mocanu, P. H. Nguyen, M. Gibescu, and A. Liotta, "A topological insight into restricted boltzmann machines," *Machine Learning*, vol. 104, no. 2-3, pp. 243–270, 2016.

[22] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 806–814.

[23] S. Narang, G. F. Diamos, S. Sengupta, and E. Elsen, "Exploring sparsity in recurrent neural networks," *CoRR*, vol. abs/1704.05119, 2017. [Online]. Available: http://arxiv.org/abs/1704.05119

[24] L. Graesser, U. Evci, E. Elsen, and P. S. Castro, "The state of sparse training in deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 7766–7792.

[25] T. Whitaker and D. Whitley, "Prune and Tune Ensembles: Low-Cost Ensemble Learning With Sparse Independent Subnetworks," Mar. 2022, arXiv:2202.11782 [cs]. [Online]. Available: http://arxiv.org/abs/2202.11782

[26] M. Cho, A. Joshi, and C. Hegde, "ESPN: Extremely Sparse Pruned Networks," in *2021 IEEE Data Science and Learning Workshop (DSLW)*, Jun. 2021, pp. 1–8.

[27] E. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239–242, Jun. 1990, conference Name: IEEE Transactions on Neural Networks.

[28] S. Dey, K.-W. Huang, P. A. Beerel, and K. M. Chugg, "Pre-defined sparse neural networks with hardware acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 332–345, 2019.

[29] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, pp. 1–12, 2018.

[30] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[31] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," Oct. 2017.

[32] S. Liu, I. Ni'Mah, V. Menkovski, D. C. Mocanu, and M. Pechenizkiy, "Efficient and effective training of sparse recurrent neural networks," *Neural Computing and Applications*, vol. 33, no. 15, pp. 9625–9636, Aug. 2021. [Online]. Available: https://rdcu.be/cegM5

[33] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *CoRR*, vol. abs/1907.04840, 2019. [Online]. Available: http://arxiv.org/abs/1907.04840

[34] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2943–2952.

[35] G. Yuan, X. Ma, W. Niu, Z. Li, Z. Kong, N. Liu, Y. Gong, Z. Zhan, C. He, Q. Jin *et al.*, "Mest: Accurate and fast memory-economic sparse training framework on the edge," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 838–20 850, 2021.

[36] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4646–4655.

[37] J. LIU, Z. XU, R. SHI, R. C. C. Cheung, and H. K. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJlbGJrtDB

[38] S. Kobayashi, S. Kiyono, J. Suzuki, and K. Inui, "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. virtual+Dublin: Association for Computational Linguistics, May 2022, pp. 42–50. [Online]. Available: https://aclanthology.org/2022.bigscience-1.4

[39] U. Evci, Y. Ioannou, C. Keskin, and Y. Dauphin, "Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, pp. 6577–6586, Jun. 2022, number: 6. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20611

[40] Z. Atashgahi, G. Sokar, T. van der Lee, E. Mocanu, D. C. Mocanu, R. Veldhuis, and M. Pechenizkiy, "Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders," *Machine Learning*, pp. 1–38, 2022.

[41] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[43] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" Oct. 2022. [Online]. Available: https://openreview.net/forum?id=Fp7__phQszn

[44] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[45] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[46] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with papi-c," in *Tools for High Performance Computing 2009*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173.

[47] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[48] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[49] S. Liu, T. V. der Lee, A. Yaman, Z. Atashgahi, D. Ferraro, G. Sokar, M. Pechenizkiy, and D. C. Mocanu, "Topological insights into sparse neural networks," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2020, pp. 279–294.

[50] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[51] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity Creation Methods: A Survey And Categorisation," *Information Fusion*, vol. 6, pp. 5–20, Mar. 2005.

[52] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, Aug. 1998, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[53] ——, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[54] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[55] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[56] Z.-H. Zhou, "Ensemble Learning," in *Machine Learning*, Z.-H. Zhou, Ed. Singapore: Springer, 2021, pp. 181–210. [Online]. Available: https://doi.org/10.1007/978-981-15-1967-3_8

[57] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, Jan. 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005800231

[58] J. Moreira, C. Soares, A. Jorge, and J. Sousa, "Ensemble Approaches for Regression: A Survey," *ACM Computing Surveys*, vol. 45, pp. 10:1–10:40, Nov. 2012.

[59] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[60] S. Dieleman and B. Schrauwen, "Accelerating sparse restricted boltzmann machine training using non-gaussianity measures," in *Deep Learning and Unsupervised Feature Learning (NIPS-2012)*, 2012.

[61] G. Sokar, E. Mocanu, D. C. Mocanu, M. Pechenizkiy, and P. Stone, "Dynamic sparse training for deep reinforcement learning," *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, 2022.

[62] S. Bibikar, H. Vikalo, Z. Wang, and X. Chen, "Federated dynamic sparse training: Computing less, communicating less, yet learning better," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, p. 6080–6088, 2022.

APPENDIX A
EXTENDED RELATED WORK

A. Ensemble Learning

Even though the underlying idea of ensembles is clear; the combination of several weak learners can yield better results than a singular learner, there is a scala of ensemble methods to realize this, varying from semi-supervised- to fully supervised ensemble methods. Literature has shown that diversity among ensemble members is key for the effectiveness of ensembles [2] [11] [12] [13] [14]. The methods to establish diversity in ensembles can be categorized as either data diversity, structural diversity, or parameter diversity. This corresponds with the three ensemble diversity technique categories described by [51]: methods affecting the starting point in hypothesis space, methods impacting the set of accessible hypotheses, and methods which alter the traversal of the hypothesis space, respectively.

A data diversity-driven and widely used supervised ensemble method is bootstrap aggregating, commonly referred to as *bagging* [2] [15] [16]. Bagging is a method which increases the data diversity of the base predictors in an ensemble by training the individual learners on replicates of the original training set which are constructed by means of bootstrapping: drawing samples with replacement. Each separate learner casts a prediction vote which are merged to a final prediction. A frequently used method to merge singular votes to a final vote is the intuitive majority-voting scheme. In line with findings for ensembles in general, literature repetitively showed the importance for high diversity among the predictors for bagging specifically as well [2] [12] [15]. The suitability of bagging for deep neural networks was challenged by [19] who found that random initialization was preferred over bagging due to the large parameter space and the need for a large training set. Even though a bootstrapped subset commonly contains the same number of instances as the original dataset [12], the subset approximately leaves out about 37% of the instances as a result of duplicates of the same instance within the bootstrapped subset [15] [19]. Yet, [15] found that increasing the size of the subsets to be twice the size of the original dataset, now only leaving out around 14% of the instances, did not improve the accuracy of the total network. Other data diversity focused ensembles include types of boosting, such as Adaptive Boosting (AdaBoost) [17] [18], and the random subspace method [52]. Whereas bagging creates replicates of the original dataset using bootstrapping, AdaBoost focuses on samples that have been misclassified by iteratively adjusting the weights of those samples [2] [17]. The random subspace method creates n decision trees, each with a randomly selected subset of features. All samples are projected into this subspace for the particular decision tree by setting the values in all other dimensions to the same value (usually zero). By doing so we in essence create n weak predictors (decision trees) which collectively outperform a singular decision tree which has all features included [52] [53]. Combining bagging and random subspace selection, in this context often also referred to as *feature bagging*, results in the state-of-the-art Random Forest network [11] [54]. Here, n random decision trees composed of a randomly selected subset of features (feature bagging) are trained on a bootstrapped subset, usually the size of the original dataset (data bagging).

Creating structural diversity in an ensemble can be done by having different model types as base predictors [2] [11]. These models can vary from simple mathematical models like linear regression to complex deep neural networks like convolutional neural networks, forming heterogeneous ensembles. The output of these individual predictors can be merged to a final vote using a method like majority voting. As presented in [55], parallel convolutional neural networks whose outputs were averaged for a final prediction, outperformed a single network. The researchers promoted ensemble diversity even further by creating replicates of the same training set by means of distortion, warping, and addition of noise. Different ensemble members were subsequently trained on the differently altered trainsets. There is however a paradoxical relation between the diversity of ensemble members and the overall accuracy of the ensemble as a whole. Both the accuracy of individual base learners and the diversity among them heavily impact the performance of the total ensemble. Yet, the more accurate the ensemble members are, the more similar they are likely to be. To find the optimal solution in this trade-off, [13] proposed the DIVACE algorithm which regularizes the training of individual ensemble members by penalizing the similarity between the models using a correlation function on top of the regular learning objective of accuracy optimization. The penalty function in this work is taken from [20]. The use of a correlation penalty is in line with findings as presented [56] entail that the generalization ability of an ensemble is dependent on the independence of the ensemble members. Despite the good results that are obtained with simpler merging methods like majority-voting schemes, averaging, or a winner-takes-all heuristics, [57] proposed a more advanced method of merging the predictions of individual ensemble:

stacked ensembles. Here, the output of the individual predictors is used as the input for the so-called *meta-learer*, the overarching predictor. We refer to an ensemble as homogeneous if for all ensemble members in the ensemble the same learning algorithm is used [58]. An ensemble with homogeneous ensemble members but a meta-learner with a different learning algorithm is considered a heterogeneous ensemble. In general, heterogeneous ensembles are expected to have higher levels of diversity [16] [58], which has shown to be crucial for the effectiveness of ensembles. Commonly, the training set for a stacked ensemble is split where one part is used to train the individual ensemble members, and the other (unseen) part is used to subsequently train the meta-learner [57].

Increasing diversity through parameter evolution is less common than the two aforementioned methods. Some parameter evolution methods are presented in [14] which include generating a 'child' base learner from two existing 'parent' base learners which achieve high accuracy, crossover operations between two base learners, and mutations of randomly selected parameters in the network. Similar to crossover operations, [19] found that parameter sharing between ensemble members, in the form of TreeNets, can outperform an ensemble with isolated base learners. The reason for this being that sharing of low-level weights allows each weight to be updated by multiple branches. Moreover, they found that the use of an ensemble-aware loss function where the gradient of the loss of the total ensemble is only back-propagated through the ensemble member with the lowest error for that sample yielded higher 'oracle' a. Here, the oracle accuracy is the accuracy of the most confident ensemble member, similar to the previously mentioned winner-takes-all scheme.

Besides these three main pillars of creating high-diversity ensembles (data-driven diversity, structural diversity, or parameter-evolution diversity), other methods include decomposition of the data (frequently used for time series data) [11] and ensembles solely based on Negative Correlation Learning as proposed by [20].

B. Sparsity

With Artificial Neural Networks finding their origin biology, it does not come as a surprise that scholars turn to biological neural networks for inspiration to improve these state-of-the-art methods even further. It was discovered by [5] that the more neurons a brain has, the fewer connections between neurons are created. Transferring this concept to ANNs showed that sparse networks could obtain the same level of accuracy as its counterpart [6]. This is in line with findings by [59] who discovered the redundancy of many of the connections in a dense network. Moreover, it was shown in [60] that many connections are valued very close to zero after training. Removing redundant weights from ANNs in a smart way seems thus a promising solution to scale deep networks.

Over the last few years, sparse implementations for several ANN architectures have been introduced among which Restricted Boltzmann Machines [21], Convolutional Neural Networks [22], and Recurrent Neural Networks [23]. The enforced sparsity levels in literature vary greatly and can range from a moderate sparsity level of 50% as in [25], to extreme sparsity levels ($>$99%) as presented in [26]. Within the field of sparsity, a distinction between two main approaches to apply sparsity to an ANN can be made: pruning of the non-critical connections of the ANN during training [27] where a dense network is the starting point (dense-to-sparse training), or a predefined level of sparsity with which an ANN is initialized after which the network is trained with a fixed topology [28] [21] or with a dynamically updated topology by pruning and regrowing weights during training [29] (sparse-to-sparse training).

1) Dense-to-sparse training: Examples of dense-to-sparse training include the works of [6], [27], and [30]. The earlier work of [27] was an initial attempt of combining the advantages of a larger neural network (i.e. being able to solve more complex problems), with those of a smaller neural network (less computational overhead and less prone to overfitting). In this work, a dense network was reduced by calculating the sensitivity of the global error to the inclusion/exclusion of each individual neuron. Sorting the neurons in order of smallest impact on the global error, the dense network was pruned up to the desired level by removing the neurons which least affected the global error. Finding a subnetwork within a dense network which equals- or even out performs a dense network was described by [6] as finding a 'winning lottery ticket'. The proposed method initializes a dense network in which a winning lottery ticket is found by iterative magnitude pruning which in turn is re-initialized and re-trained. Even though the re-training of the pruned network is significantly faster than the original, the computational overhead of the initial dense network should not be overlooked, especially when considering extremely large networks. Despite a wide variety of approaches of dense-to-sparse training methods, all have in common that the computational overhead is equal or more than a regular dense network.

2) Sparse-to-sparse training: In contrast, a sparse-to-sparse training regime starts with a sparsely initiated network which is subsequently trained. The odds of the randomly sparsely initialized network to be one of the subnetworks which can equal- or even outperform its dense counterpart, is very slim. A network should thus be able to train and evaluate many different topologies during training. In recent years, the concept of dynamic sparse training as introduced by [29] has gained a lot of popularity among scholars. The proposed SET algorithm evolves an initially sparse topology to a sparse network reaching high levels of accuracy by pruning weights based on magnitude and randomly regrow the same amount of weights that were just pruned. This method achieved better performance than its dense counterparts or static sparse neural networks that were trained from scratch [29]. The effectiveness of adaptive sparse connectivity was confirmed by [49] who were able to visualize the topological optimization process of SET. The usage of the SET procedure has expanded beyond the networks as demonstrated by the original authors and alternative applications such as deep reinforcement learning as presented by [61] are out there. A more elaborate overview of applications of sparse learning in the context of deep reinforcement learning is presented in [24]. A big advantage of methods like the SET procedure [29], among which the sparse training of RNNs (ST-RNN) as presented by [32], Sparse Momentum [33], Rigged Lottery (RigL) [34], Memory-Economic Sparse training [35], and dynamic reparametrization [36], is that the network is sparsely initialized in contrast to the aforementioned methods like the lottery ticket hypothesis by [6] or earlier works like [27]. Improving efficiency even further, [62] proposed the federated dynamic sparse training protocol which entailed applying a dynamic sparse training procedure on C distributed Clients which where trained on samples of the original dataset. [37] proposed a method including trainable masks, leaving the original weight matrix untouched when pruning during training, and by doing so preserving the historical information about the parameter importance.

C. Sparse ensembles

At the intersection of ensemble learning and sparsity some work has been done. Continuing on the work of [6], Kobayashi et al. used iterative magnitude pruning to create an ensemble where a subnetwork was stored at each iteration of pruning [38]. A more recent work by [25] created ensemble members by copying a dense parent network and prune and tune each network individually. Both of these methods however, are dense-to-sparse training regimes, meaning that the potential benefit of sparse ensembles in terms of computational overhead is not achieved. The earlier discussed work of [62] in the field of Federate learning could be considered a successful sparse-to-sparse ensemble with a meta-learner. All individual submodels are remotely, but sparsely, trained and updated. A recently published work by [10] presented two methods with no computational overhead during either training or testing. The first method, Dynamic Sparse Training (DST) Ensemble, initializes each sparse ensemble member network and subsequently trains this network using the Rigged Lottery method as presented in [34]. The second proposed method, Efficient Dynamic Sparse Training (EDST) Ensemble, initializes only one sparse network, hence the efficiency. This network is initially trained using a relatively large learning rate, exploring a large range of the parameter solution space (exploration phase). Subsequently, the network is refined from the previous network using a smaller learning rate. Once the network has converged, the current state is stored as a subnetwork. The bassin in the solution space in which the network converged is escaped by pruning a large percentage of the weights with the lowest magnitude (refinement phase). The refinement phase is repeated for a predetermined number of iterations, collecting M free tickets. [39] found 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. The SET procedure seemed to be less effective at this. They also found that the Lottery Ticket Hypothesis as presented by [6] does not improve gradient flow in earlier or later training stages. Moreover, they showed that the lottery tickets initialization remains within the same basin in the solution space as the pruning solution, making the lottery tickets fundamentally limited in their ability to improve training of sparse neural networks [39]. This might be explained by the resulting lack of diversity in an ensemble as explained in Section A-A.

Most research in the field of sparse training is mostly focused on studying the effect of sparsity in artificial neural networks. However, even though the effects of sparsity are successfully captured, practical application to reduce the computational overhead is mostly neglected: the sparsity in most networks is enforced by applying binary masks to the weight matrices. The reason for this mostly being the lack of sparse linear algebra support.

D. Truly Sparse Training

Little work has been done in the field of 'truly' sparse training. With truly sparse training we refer to the training of neural networks where we do not use dense matrices which mostly contain uselessly zero-valued weights. Almost all work in the field of sparse network makes use of binary masked weight matrices which still cause significant computational overhead.

Novel results were presented by [8] who created a truly sparse implementation of the SET algorithm for Multi-Layer Perceptrons. The impact of a truly sparse implementation versus a mask-enforced sparse network becomes evident as the truly sparse implementation allows an MLP with hundreds of thousands of neurons to be trained on a regular laptop without GPU support. [9] continued on this work and introduced a parallel training algorithm for truly sparse networks and a new method called *Importance pruning* to reduce the number of parameters even further.

APPENDIX B
DISTANCE IMPLEMENTATION ALGORITHM

Algorithm 3: Truly Sparse EDST 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 ← 0 to k do
    W_i ← ErdosRenyiInit(W_i)
    b_i ← ZeroVector()
end
globalExploration ← CalculateGlobalPruningEpochs(numEpochs)
%Training
for epoch ← 0 to numEpochs do
    model.train_step()
    if epoch % e == 0 then
        model.SET_weight_evolution(p)
    if epoch in globalExploration then
        for i ← 0 to k − 1 do
            if i == 0 then
                W_i.SET_weight_evolution(q)
            else
                W_i' ← PruneSmallestK(|W_i|, q)
                for l ← 0 to 100 do
                    totalDistance +=
                    CalculateDistance(W_i, W_i' ∪ RegrowRandomK(q))
                end
                μ ← GetAverage(totalDistance)
                σ ← GetStandardDeviation(totalDistance)
                currentDistance ← 0
                iteration ← 0
                while currentDistance < ( μ + λ × σ) do
                    if iteration > maxIterations then
                        Throw Iteration Error and stop training
                    proposedWeights ← W_i' ∪ RegrowRandomK(q)
                    currentDistance ← CalculateDistance(W_i, proposedWeights)
                    iteration += 1
                end
                W_i ← W_i' ∪ proposedWeights
        end
end
```

APPENDIX C
DISJOINT IMPLEMENTATION ALGORITHM

Algorithm 4: Truly Sparse EDST 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 ← 0 to k do
    W_i ← ErdosRenyiInit(W_i)
    b_i ← ZeroVector()
    Blocked_i ← ∅
end
globalExploration ← CalculateGlobalPruningEpochs(numEpochs) %Training
for epoch ← 0 to numEpochs do
    model.train_step()
    if epoch % e == 0 then
        model.SET_weight_evolution(p)
    if epoch in globalExploration then
        for i ← 0 to k − 1 do
            if i == 0 then
                W_i.SET_weight_evolution(q)
            else
                for j ← 0 to W_i.numRows() do
                    Blocked_i.add(tuple(W_i.row[j] , W_i.col[j]))
                end
                W_i' ← PruneSmallestK(|W_i|, q)
                numWeightsRegrown ← 0
                iteration ← 0
                while numWeightsRegrown ≠ K do
                    proposedWeights ← (RegrowRandomK(q) - numWeightsRegrown)
                    for j ← 0 to W_i.numRows() do
                        if iteration > maxIterations then
                            store(proposedWeights.numRows())
                            break
                        else if tuple(proposedWeights.row[j], proposedWeights.col[j]) in Blocked_i then
                            proposedWeights.delete(j)
                    end
                    W_i' ← W_i' ∪ proposedWeights
                    numWeightsRegrown += proposedWeights.numRows()
                    iteration += 1
                end
                W_i ← W_i' ∪ proposedWeights
        end
end
```

APPENDIX D
EXPERIMENTAL RESULTS CIFAR-10, GP & HIGGS

TABLE VII: Summary of experiments of our EDST implementations and baselines on the CIFAR-10 dataset. 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.

Architecture	**Model**	**Sparsity**	**Results**			
			Accuracy [%]	**Weights [#]**	**Train Flops [#]**	**Train time [min]**
3072-1000-1000-1000-10	Single Dense Model	-	57.0	5,085,010	2.18e14	$\sim$ 693.1
	EDST	0.98	**62.8**	0.08x	0.03x	$\sim$ 138.3
		0.97	62.0	0.16x	0.05x	$\sim$ 178.3
		0.95	61.8	0.24x	0.08x	$\sim$ 290.9
	RD-EDST (ours)	0.98	61.6	0.08x	0.03x	$\sim$ 119.7
		0.97	61.5	0.16x	0.05x	$\sim$ 230.1
		0.95	61.6	0.24x	0.08x	$\sim$ 275.7
	FD-EDST (ours)	0.98	61.3	0.08x	0.03x	$\sim$ 135.2
		0.97	60.7	0.16x	0.05x	$\sim$ 231.3
		0.95	60.6	0.24x	0.08x	$\sim$ 312.9
	D-EDST ($\lambda = 1.0$) (ours)	0.98	62.5	0.08x	0.03x	$\sim$ 119.7
		0.97	61.8	0.16x	0.05x	$\sim$ 173.8
		0.95	61.6	0.24x	0.08x	$\sim$ 287.9
	D-EDST ($\lambda = 1.5$) (ours)	0.98	62.6	0.08x	0.03x	$\sim$ 121.5
		0.97	61.0	0.16x	0.05x	$\sim$ 225.0
		0.95	61.9	0.24x	0.08x	$\sim$ 282.3
	D-EDST ($\lambda = 2.0$) (ours)	0.98	62.2	0.08x	0.03x	$\sim$ 121.8
		0.97	61.6	0.16x	0.05x	$\sim$ 223.8
		0.95	61.5	0.24x	0.08x	$\sim$ 277.2
	SET-MLP	0.97	59.2	0.03x	0.05x	$\sim$ 216.4
	Single Static Sparse Model	0.97	57.1	0.03x	0.05x	$\sim$ 206.4
	Dense Ensemble	-	61.0	5.00x	5.00x	$\sim$ 3234.2

TABLE VIII: Summary of experiments of our EDST implementations and baselines on the Gesture Phase Segmentation dataset.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.

Architecture	**Model**	**Sparsity**	**Results**			
			Accuracy [%]	**Weights [#]**	**Train Flops [#]**	**Train time [min]**
50-1000-1000-1000-5	Single Dense Model	-	54.9	2,058,005	1.71e13	$\sim$ 60.0
	EDST	0.98	71.8	0.12x	0.04x	$\sim$ 16.8
		0.95	72.2	0.25x	0.07x	$\sim$ 24.6
		0.93	72.5	0.37x	0.11x	$\sim$ 33.0
	RD-EDST (ours)	0.98	71.8	0.12x	0.04x	$\sim$ 17.1
		0.95	72.6	0.25x	0.07x	$\sim$ 31.1
		0.93	72.9	0.37x	0.11x	$\sim$ 53.1
	FD-EDST (ours)	0.98	71.6	0.12x	0.04x	$\sim$ 18.1
		0.95	74.2	0.25x	0.07x	$\sim$ 36.9
		0.93	**75.3**	0.37x	0.11x	$\sim$ 74.0
	D-EDST ($\lambda = 1.0$) (ours)	0.98	72.7	0.12x	0.04x	$\sim$ 16.6
		0.95	73.4	0.25x	0.07x	$\sim$ 25.3
		0.93	73.3	0.37x	0.11x	$\sim$ 32.5
	D-EDST ($\lambda = 1.5$) (ours)	0.98	72.0	0.12x	0.04x	$\sim$ 16.2
		0.95	73.4	0.25x	0.07x	$\sim$ 25.7
		0.93	73.5	0.37x	0.11x	$\sim$ 32.2
	D-EDST ($\lambda = 2.0$) (ours)	0.98	72.2	0.12x	0.04x	$\sim$ 16.8
		0.95	73.9	0.25x	0.07x	$\sim$ 25.1
		0.93	73.7	0.37x	0.11x	$\sim$ 33.1
	SET-MLP	0.95	60.4	0.05x	0.07x	$\sim$ 25.6
	Single Static Sparse Model	0.95	67.4	0.05x	0.07x	$\sim$ 22.7
	Dense Ensemble	-	54.0	5.00x	5.00x	$\sim$ 299.9

TABLE IX: Summary of experiments of our EDST implementations and baselines on the HIGGS dataset. 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.

Architecture	**Model**	**Sparsity**	**Results**			
			Accuracy [%]	**Weights [#]**	**Train Flops [#]**	**Train time [min]**
28-1000-1000-1000-2	Single Dense Model	-	57.1	2,033,002	2.28e14	$\sim$ 709.6
	EDST	0.98	53.1	0.12x	0.04x	$\sim$ 241.6
		0.95	53.1	0.25x	0.07x	$\sim$ 330.1
	RD-EDST (ours)	0.98	64.3	0.12x	0.04x	$\sim$ 222.7
		0.95	63.5	0.25x	0.07x	$\sim$ 316.5
	FD-EDST (ours)	0.98	64.0	0.12x	0.04x	$\sim$ 229.1
		0.95	63.7	0.25x	0.07x	$\sim$ 339.2
	D-EDST ($\lambda = 1.0$) (ours)	0.98	64.4	0.12x	0.04x	$\sim$ 229.0
		0.95	63.6	0.25x	0.07x	$\sim$ 319.4
	D-EDST ($\lambda = 1.5$) (ours)	0.98	**64.5**	0.12x	0.04x	$\sim$ 232.0
		0.95	63.5	0.25x	0.07x	$\sim$ 343.0
	D-EDST ($\lambda = 2.0$) (ours)	0.98	64.2	0.12x	0.04x	$\sim$ 238.9
		0.95	63.7	0.25x	0.07x	$\sim$ 322.5
	SET-MLP	0.95	54.3	0.05x	0.07x	$\sim$ 332.1
	Single Static Sparse Model	0.95	53.1	0.05x	0.07x	$\sim$ 315.4
	Dense Ensemble	-	60.4	5.00x	5.00x	$\sim$ 3561.5

APPENDIX E
EXPERIMENT HYPERPARAMETERS

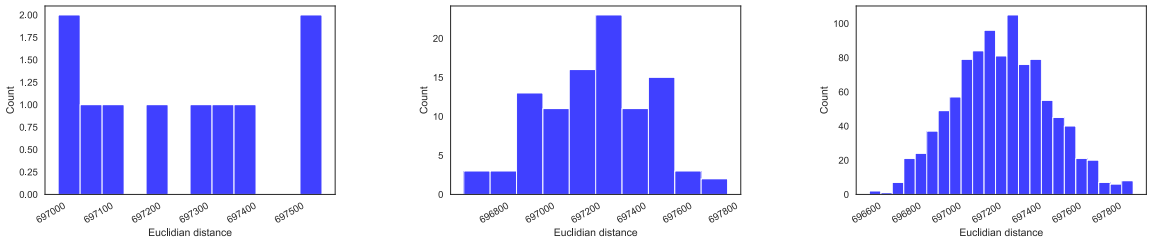
Given the very limited availability of literature implementing the truly sparse framework, we mostly used the same configuration of hyperparameters as presented in [9]. For all training, we used Stochastic Gradient Descent with a momentum of 0.9 and a weight decay of 0.0002. In contrast to the works of [9] and [29], we use a slightly lower dropout rate of 0.2 as our experiments concern very sparse networks. An overview of the main hyperparameters can be found in Table X. We used the Alternated Left ReLU activation 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 literature. We thus selected a neutral value of 0.5. For the aggregated datasets we instead applied the regular Rectified Linear Unit activation function as selecting a single value for α for all datasets yielded relatively poor overall performance. All models were trained with a batch size of 128. The learning rate η and evolution frequency e varied for the EDST 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 planned to run all experiments with a fixed learning rate of 0.01 as presented in [29]. However, due to the numerical instability of the truly sparse framework, we had to decrease this learning rate and used a fixed learning rate of 0.001 for HIGGS and the Gesture Phase Segmentation dataset. Due to the high number of input features for CIFAR-10, we were forced to use a fixed learning rate of 0.0001. All ensembles, except for those in Section IV-G, consist of 5 subnetworks.

TABLE X: Table of hyperparameters used for the experiments.

Experiment	**Dataset**	**Architecture**	**Hyperparameters**				
			η	e	$\mathcal{A}$	α	**Weight init.**
EDST Ensembles	CIFAR-10	3072x1000x1000x1000x10	0.1 - 0.05	2-4-8	All-ReLU	0.75	He Uniform
	Gesture Phase Segmentation	50x1000x1000x1000x5	0.1 - 0.05	2-4-8	All-ReLU	0.5	He Uniform
	HIGGS	28x1000x1000x1000x2	0.1 - 0.05	2-4-8	All-ReLU	0.05	Xavier
	Aggregated datasets	...x1000x1000x1000x2	0.1 - 0.05	2-4-8	ReLU	-	He Uniform
Baseline Ensembles	CIFAR-10	3072x1000x1000x1000x10	0.01	2	All-ReLU	0.75	He Uniform
	Gesture Phase Segmentation	50x1000x1000x1000x5	0.01	2	All-ReLU	0.5	He Uniform
	HIGGS	28x1000x1000x1000x2	0.01	2	All-ReLU	0.05	Xavier
	Aggregated datasets	...x1000x1000x1000x2	0.01	2	ReLU	-	He Uniform
Dense Models	CIFAR-10	3072x1000x1000x1000x10	0.0001	-	ReLU	-	He Uniform
	Gesture Phase Segmentation	50x1000x1000x1000x5	0.001	-	ReLU	-	He Uniform
	HIGGS	28x1000x1000x1000x2	0.001	-	ReLU	-	Xavier
	Aggregated datasets	...x1000x1000x1000x2	0.001	-	ReLU	-	He Uniform

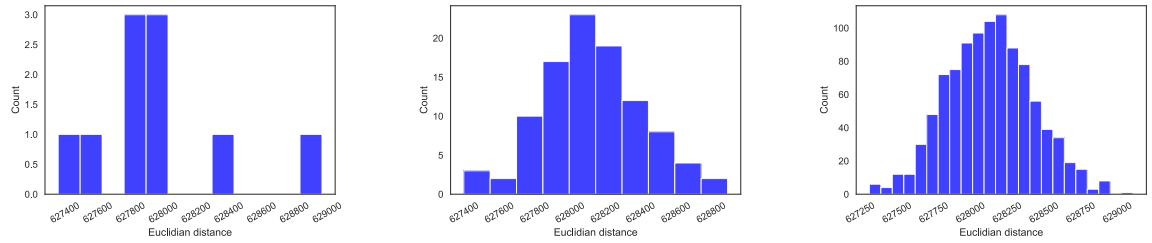
APPENDIX F
EUCLIDEAN DISTANCE DISTRIBUTION AFTER GLOBAL PRUNING

In this appendix, we visualize the distribution of the euclidean distance between converged subnetworks and randomly newly re-initialized networks. More specifically, we calculate the euclidean distance between the weight matrix of a converged subnetwork's hidden layer and the weight matrix after pruning- and regrowing 80% of the weights. If we define W_i to be the hidden layer of the converged subnetwork, W_i' as the matrix we get after pruning 80% of the weights (based on magnitude), and W_i^{random} as the random weights we regrow to replace the pruned weights, we obtain the weight matrix of the newly initialized subnetwork W_i^{new} as follows: $W_i^{new} = W_i' \cup W_i^{random}$. Here, we randomly generate 10, 100, or a thousand W_i^{new} weight matrices and visualize the euclidean distance to W_i for CIFAR-10, the Gesture Phase Segmentation dataset, and HIGGS. It becomes clear from Figures 5, 6, and 7 that the distribution of euclidean distances starts approximating a gaussian distribution from around 100 iterations.



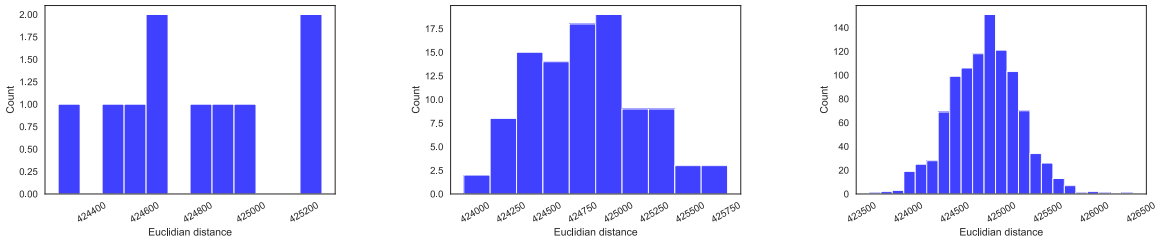
(a) Ten iterations (b) Hundred iterations (c) Thousand iterations

Fig. 5: Visualization of the euclidean distance distribution on CIFAR-10.



(a) Ten iterations (b) Hundred iterations (c) Thousand iterations

Fig. 6: Visualization of the euclidean distance distribution on the Gesture Phase Segmentation dataset.



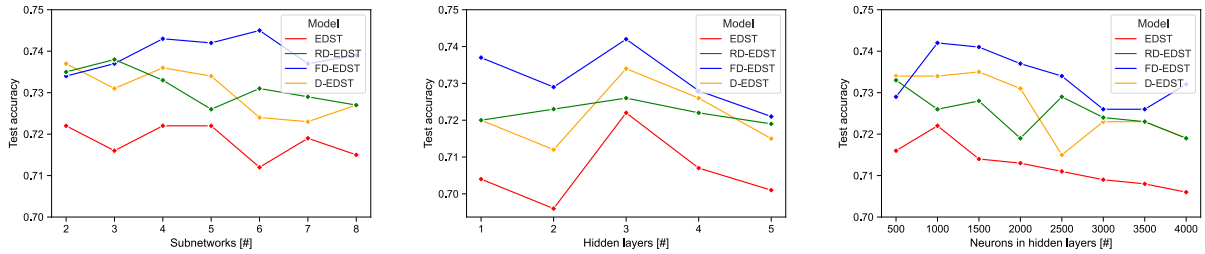
(a) Ten iterations (b) Hundred iterations (c) Thousand iterations

Fig. 7: Visualization of the euclidean distance distribution on the HIGGS dataset.

APPENDIX G
SUBNETWORK COUNT, DEPTH, AND WIDTH

We evaluated the various models on the Gesture Phase Segmentation dataset for a varying number of subnetworks, hidden layers, and number of neurons per hidden layer. Figure 8 provides an overview of the obtained results. For the experiments, we used the same configuration as described in Section IV-C unless indicated differently. The results presented in Figure 8a were obtained by varying the number of subnetworks in the ensemble but keeping all other hyperparameters, including the total number of epochs, the same. This means that with an increase in the number of subnetworks, the number of training epochs per subnetwork decreases. We observe in Figure 8a that there seems to be a slight decreasing trend in terms of accuracy for all models except FD-EDST. Yet, more experiments are required to determine the validity of this observation.

For Figures 8b and 8c, we see that the pattern of the relation between the depth and width of the subnetworks, and test accuracy, is almost the same for all implementations. This might suggest that the networks suffer from poor generalization and/or overparameterization independent of the diversification method.



(a) (b) (c)

Fig. 8: Lineplots of the performance of all the models in terms of classification accuracy on the Gesture Phase Segmentation test set with a varying number of submodels (8a), hidden layers (8b), and number of neurons per hidden layer (8c).

APPENDIX H
AGGREGATED DATASETS

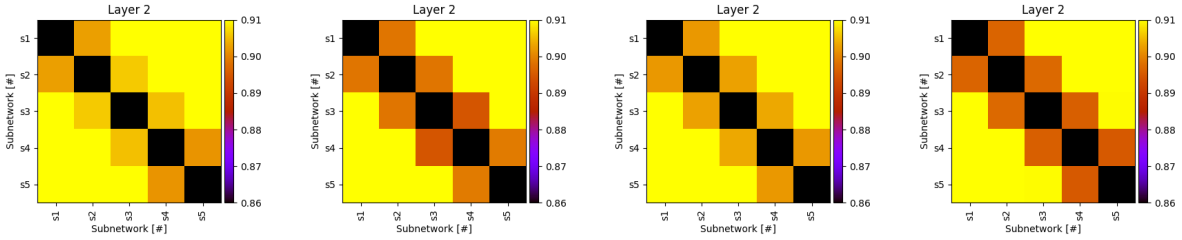
For all datasets, the models were initialized with the *He Uniform* initializer, and an 80-20 train-split was used to obtain separate sets for training and evaluation. An overview of the data properties for all datasets can be found in Table XI. All datasets were transformed to binary classification problems by [43]. Given our computational- and time constraints, the HIGGS- and covertype datasets were removed from the original collection of datasets due to their sizes.

TABLE XI: Summary of aggregated datasets

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

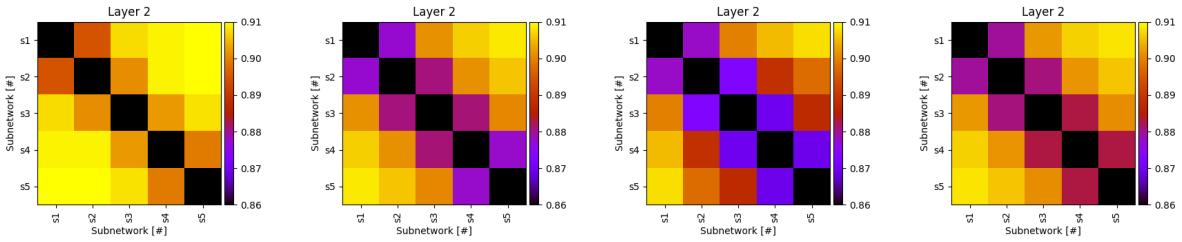
APPENDIX I
FINAL SUBNETWORK TOPOLOGY

We visualize the topological distances between the converged subnetworks. We do this for CIFAR-10 (9), the Gesture Phase Segmentation dataset (10), and the HIGGS dataset (10). The topological distances of the subnetworks of our proposed models that were trained on the Gesture Phase Segmentation dataset are smaller than the original implementation. Further research is required to discover why this only happens for this particular dataset. For the other two datasets, the topological distance of the converged subnetworks is similar to the original implementation.



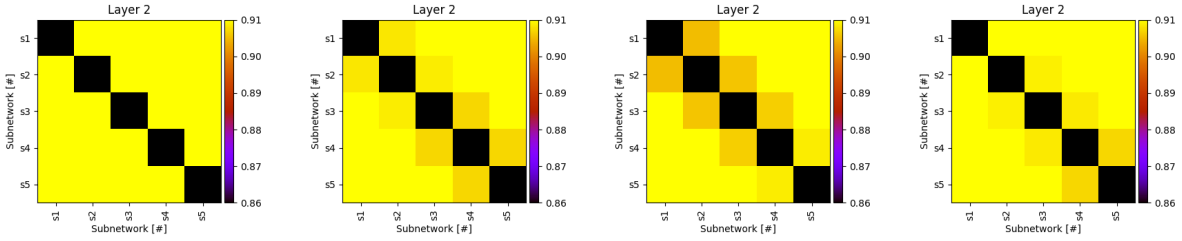
(a) EDST (b) Regular Disjoint EDST (c) Fully Disjoint EDST (d) Distance EDST ($\lambda = 1.5$)

Fig. 9: Topological distance of converged subnetworks that were trained on CIFAR-10.



(a) EDST (b) Regular Disjoint EDST (c) Fully Disjoint EDST (d) Distance EDST ($\lambda = 1.5$)

Fig. 10: Topological distance of converged subnetworks that were trained on Gesture Phase Segmentation dataset



(a) EDST (b) Regular Disjoint EDST (c) Fully Disjoint EDST (d) Distance EDST ($\lambda = 1.5$)

Fig. 11: Topological distance of converged subnetworks that were trained on HIGGS.

APPENDIX J
ABLATION STUDY COMPREHENSIVE REFINEMENT

In this appendix, we present all the results from the ablation study into the impact of our Comprehensive Refinement phase, including the number of training Floating Point Operations and total training time in minutes. From Table XII we see that our Comprehensive refinement phase does not cause any computational overhead in terms of training FLOPs. The training times differ for some models but this applies both to the runs with just learning rate reduction and our proposed Comprehensive Refinement phase. We assume that these differences are the result of the stochastic nature of the regrowing process of the SET procedure [29].

TABLE XII: Ablation study Comprehensive Refinement phase on CIFAR-10, the Gesture Phase Segmentation (GPS) dataset, and HIGGS dataset.

Dataset	**Model**	**Comprehensive Refinement**	**Results**		
			Accuracy [%]	**Train Flops [#]**	**Train time [min]**
CIFAR-10	RD-EDST (ours)	No	62.1	1.15e13	$\sim$ 222.1
		Yes	61.5	1.15e13	$\sim$ 230.1
	FD-EDST (ours)	No	61.0	1.15e13	$\sim$ 251.2
		Yes	60.7	1.14e13	$\sim$ 231.3
	D-EDST ($\lambda = 1.5$) (ours)	No	**62.3**	1.15e13	$\sim$ 210.2
		Yes	61.0	1.15e13	$\sim$ 225.0
Gesture Phase Segmentation	RD-EDST (ours)	No	71.2	1.27e12	$\sim$ 31.7
		Yes	72.6	1.26e12	$\sim$ 31.1
	FD-EDST (ours)	No	73.9	1.27e12	$\sim$ 57.4
		Yes	**74.2**	1.26e12	$\sim$ 36.9
	D-EDST ($\lambda = 1.5$) (ours)	No	71.7	1.28e12	$\sim$ 26.4
		Yes	73.4	1.28e12	$\sim$ 25.7
HIGGS	RD-EDST (ours)	No	**63.7**	1.65e13	$\sim$ 338.3
		Yes	63.5	1.65e13	$\sim$ 316.5
	FD-EDST (ours)	No	63.5	1.65e13	$\sim$ 358.3
		Yes	**63.7**	1.65e13	$\sim$ 339.2
	D-EDST ($\lambda = 1.5$) (ours)	No	63.5	1.65e13	$\sim$ 319.0
		Yes	63.5	1.65e13	$\sim$ 343.0