Supervised Feature Selection via Ensemble Gradient Information from Sparse Neural Networks

Kaiting Liu[1] **Zahra Atashgahi**[2] **Ghada Sokar**[1]
Mykola Pechenizkiy[1] **Decebal Constantin Mocanu**[3,1]
[1]Eindhoven University of Technology [2]University of Twente [3]University of Luxembourg

Abstract

Feature selection algorithms aim to select a subset of informative features from a dataset to reduce the data dimensionality, consequently saving resource consumption and improving the model's performance and interpretability. In recent years, feature selection based on neural networks has become a new trend, demonstrating superiority over traditional feature selection methods. However, most existing methods use dense neural networks to detect informative features, which requires significant computational and memory overhead. In this paper, taking inspiration from the successful application of local sensitivity analysis on neural networks, we propose a novel resource-efficient supervised feature selection algorithm based on sparse multi-layer perceptron called "GradEnFS". By utilizing the gradient information of various sparse models from different training iterations, our method successfully detects the informative feature subset. We performed extensive experiments on nine classification datasets spanning various domains to evaluate the effectiveness of our method. The results demonstrate that our proposed approach outperforms the state-of-the-art methods in terms of selecting informative features while saving resource consumption substantially. Moreover, we show that using a sparse neural network for feature selection not only alleviates resource consumption but also has a significant advantage over other methods when performing feature selection on noisy datasets.

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

1 INTRODUCTION

With the increasing dimensionality of the real-world data, training a machine learning model on these high-dimensional data brings several limitations, including the curse of dimensionality, overfitting, and expensive resource consumption. In the face of these challenges, feature selection remains a crucial step in the machine learning pipeline. It aims to identify the most relevant and informative features from a dataset while removing irrelevant, noisy, or redundant ones. By reducing data dimensionality and mitigating noise, feature selection enhances the efficiency of the model's training phase and the effectiveness and interpretability of the trained models (Li et al., 2018).

Numerous feature selection algorithms have been proposed, and there is growing interest in using neural networks for feature selection due to their ability to learn non-linear dependencies between features effectively (Lemhadri et al., 2021; Yamada et al., 2020). However, most existing neural network-based (NN-based) feature selection methods suffer from the demand for substantial computational and memory resources because they often require training dense neural networks, which are highly overparameterized, for many iterations to identify informative features. This resource-intensive characteristic hinders their scalability on the super high-dimensional dataset and low-resource environment. The emergence of Dynamic Sparse Training (DST) algorithms, capable of training sparse neural networks from scratch, offers a promising solution to this challenge. Novel feature selection methods that leverage the sparse networks trained by DST algorithms have been presented, effectively mitigating the resource consumption issue (Atashgahi et al., 2020; Sokar et al., 2022). QuickSelection (QS) (Atashgahi et al., 2020) is the pioneering method that utilizes a sparse network trained by DST to successfully identify informative features. It reduces memory and computational costs compared to previous dense-based methods. However, QS exhibits slow conver-

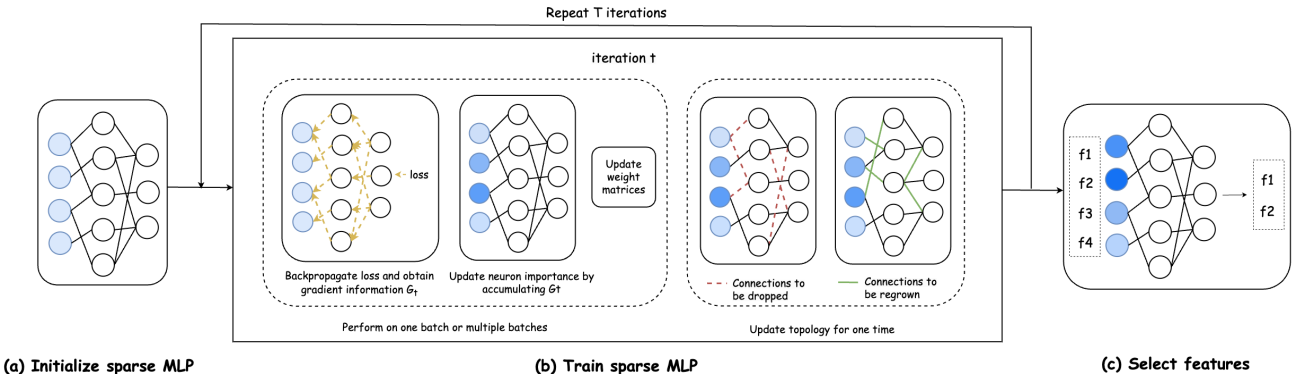

Figure 1: An overview of our proposed method "GradEnFS". GradEnFS first initializes a sparse MLP and neuron importance for all input neurons. During training, the algorithm performs backpropagation and accumulates the gradient information to update neuron importance. Meanwhile, the weight matrices and topology are also updated. After training, input neurons with the highest importance are selected as the most informative features.

gence and needs numerous training epochs for informative feature detection. WAST (Sokar et al., 2022) is another sparse NN-based method that addresses the limitations of QS and exhibits fast convergence. However, it is specifically tailored for unsupervised contexts.

In this paper, we propose an efficient supervised feature selection algorithm based on sparse Multi-Layer Perceptrons (MLPs). Our approach draws inspiration from the local sensitivity method, which is used to analyze the functionality of components within a function (Zhou and Lin, 2008). Specifically, we present a metric that leverages gradient information to quantify the importance of neurons during the training of a sparse MLP using the DST algorithm. It utilizes the gradient information of models from different training iterations, resulting in an implicit ensemble effect. Therefore, we named this method **Grad**ient **En**semble **F**eature **S**election (GradEnFS). We evaluate our proposed approach on nine datasets from various domains and demonstrate that it outperforms state-of-the-art feature selection methods in balancing the performance of feature selection tasks and resource consumption considerations. Our main contributions are:

- We introduce a novel sensitivity-based neuron importance metric and use it to propose GradEnFS, a highly efficient sparse neural network-based supervised feature selection algorithm.
- Our extensive evaluation of GradEnFS across nine benchmark datasets spanning diverse domains demonstrates its superiority over state-of-the-art techniques in selecting informative features.
- GradEnFS significantly reduces memory usage, computational cost, and the number of convergence iterations compared to other neural network-based methods.
- We highlight GradEnFS's remarkable efficiency in handling noisy datasets, showcasing its robustness and effectiveness toward noisy datasets.

2 BACKGROUND & RELATED WORK

2.1 Feature Selection

Feature selection generally is classified into three categories: filter, wrapper, and embedded method. The **filter** method selects the most informative feature by utilizing a statistic-based score independent of the downstream learning task. Two famous examples of such scores are mutual information (Lee et al., 2012) and F-score (Ding, 2009). The filter approach is simple to design and does not require significant computing resources, which are advantages when working with large datasets. However, they are not able to learn the non-linear relation among features and are prone to choose redundant features (Chandrashekar and Sahin, 2014). The **wrapper** method evaluates the score of the features' subsets via the accuracy of a specific model. For example, Maldonado and Weber (2009) proposed a wrapper method using a support vector machine. This method requires retraining the model per subset, which brings expensive computational costs, especially for high-dimensional data, even with a simple model (El Aboudi and Benhlima, 2016). The **embedded** method performs feature selection si-

multaneously in the model's learning phases to alleviate the problem of filter and wrapper methods, and a successful example is LASSO (Tibshirani, 1996).

Recently, one of the branches in the embedded method, the NN-based feature selection, has become a new trend due to some attractive advantages of neural networks, such as the ability to learn the non-linear relation between features (Lemhadri et al., 2021; Lu et al., 2018; Yamada et al., 2020; Abid et al., 2019; Han et al., 2018). However, most of these methods suffer from overparameterization, which leads to expensive resource costs, especially when facing high-dimensional datasets (Singh et al., 2020). The newly emerged algorithms that utilize sparse neural networks for feature selection offer a promising solution for addressing the resource-intensive issue. QS is the first sparse NN-based feature selection method (Atashgahi et al., 2020). It introduces the strength of the neuron, which also refers to neuron importance, trains a sparse autoencoder from scratch, and subsequently derives the ranking of the informative features according to the designed neuron importance metric at the end of the training. WAST algorithm is another sparse NN-based method recently proposed (Sokar et al., 2022). WAST introduces a novel neuron importance metric and develops a new sparse training algorithm that incorporates this metric in drop-and-regrow cycles. This approach enables the network's topology to focus on informative neurons and connections, rapidly detecting the most informative input neurons (features). However, QS exhibits slow convergence, and WAST is unsuitable for supervised contexts.

2.2 Sparse Neural Networks

Sparse Neural Networks (SNNs) are neural networks that encourage connection sparsity in the network's topology (Mocanu et al., 2021). SNNs offer potential solutions to address the resource-intensive demands and overfitting issues arising from the overparameterization of Dense Neural Networks (DNNs), where neurons in DNNs are connected in pairs between consecutive layers. The first SNNs were obtained through **pruning methods**, which involve removing connections from a trained DNN based on specific criteria (Janowsky, 1989; Mozer and Smolensky, 1989; LeCun et al., 1989; Han et al., 2015; Frankle and Carbin, 2018; Strom, 2015; Molchanov et al., 2019, 2016; Louizos et al., 2017; Wen et al., 2016; Srinivas et al., 2017; Zhu and Gupta, 2017; Lee et al., 2018; Wang et al., 2020; Tanaka et al., 2020; de Jorge et al., 2020). However, pruning methods necessitate training a DNN first, which means the benefits of sparsity are only realized during the model's inference phase. **Dynamic Sparse Training** (DST) is a recent and promising approach for obtaining SNNs, attracting significant research interest (Mocanu et al., 2017, 2018; Dettmers and Zettlemoyer, 2019; Evci et al., 2020; Dai et al., 2019; Atashgahi et al., 2022). DST methods initialize a sparse neural network and, during training, periodically update the sparse topology through a drop-and-grow cycle. This cycle involves dropping a fraction of connections based on a dropping criterion and regrowing the same number of connections based on a regrown criterion. Because DST methods train a sparse neural network from scratch, the benefits of the sparsity can be obtained in both the training and inference phases. DST methods have demonstrated success in diverse research fields, such as continual learning (Sokar et al., 2021a), feature selection (Atashgahi et al., 2020; Sokar et al., 2022), federated learning (Zhu and Jin, 2019; Bibikar et al., 2022) and deep reinforcement learning (Sokar et al., 2021b).

2.3 Local Sensitivity Analysis

Local Sensitivity Analysis (LSA) is a widely used method to measure the importance of individual input features in machine learning models, including neural networks. It quantifies the local impact of each input feature on the model's output at a specific sample by calculating the first-order partial derivative of the output with respect to the input. LSA defines this derivative as the corresponding input's local sensitivity and can be expressed using the following formula:

$$s_j(\boldsymbol{x}^{(i)}) = \frac{\partial y^{(i)}}{\partial x_j}(\boldsymbol{x}^{(i)}) \tag{1}$$

where $\boldsymbol{x}^{(i)}$ represents the specific i-th sample, $y^{(i)}$ corresponds to the model's output of this input sample, x_j denotes the j-th feature in the input vector, and $s_j(\boldsymbol{x}^{(i)})$ represents the local sensitivity that quantifies how a slight change in the value of the j-th feature of the input vector $\boldsymbol{x}^{(i)}$ will affect the output.

Based on this principle, several applications have been proposed to analyze the relationship between input features and output in deep learning models, especially in the analysis of convolutional neural networks (Simonyan et al., 2013; Smilkov et al., 2017; Selvaraju et al., 2017). However, local sensitivity calculated based on a single sample can easily introduce bias. Hence, aggregating local sensitivity values obtained from a wide range of samples using methods such as averaging provides a better solution (Pizarroso et al., 2020).

3 METHODOLOGY

3.1 Problem Formulation

Let $\mathbb{D}$ be a dataset containing m samples $\{\boldsymbol{x}^{(i)}, y^{(i)}\}$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is the i-th sample in the data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times d}$ with the dimensionality of the features d, and $y^{(i)}$ is the label corresponding to the $\boldsymbol{x}^{(i)}$. Feature selection algorithm aims to select the most informative feature subset $\mathbb{F}_s$ from the whole feature space $\mathbb{F}$, and $|\mathbb{F}_s| = K$, where K is the hyperparameter of the algorithm which notates the number of the selected features. The objective function of the feature selection problem in a supervised learning context is as follows:

$$\mathbb{F}_s^* = \underset{\mathbb{F}_s \subset \mathbb{F}, |\mathbb{F}_s|=K}{\operatorname{argmin}} \sum_{i=1}^{m} \mathcal{L}(f(\boldsymbol{x}_{\mathbb{F}_s}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \tag{2}$$

where $f(\boldsymbol{x}_{\mathbb{F}_s}^{(i)}; \boldsymbol{\theta})$ represents a function with parameter $\boldsymbol{\theta}$ that predicts the target of the i-th sample utilizing the feature subset $\mathbb{F}_s$, $\mathcal{L}$ is the loss function, and $\mathbb{F}_s^*$ is the optimal feature subset which minimize the loss. Notably, determining the optimal feature subset $\mathbb{F}^*$ is an NP-hard problem because the number of potential feature subsets increases exponentially with the dimensionality of features d. Therefore, most existing algorithms exploit heuristic sequential search to adjust subsets iteratively instead of directly optimizing Eq(2); for example, some works rely on importance metrics to rank the features. Our method follows this approach, which utilizes the importance metric based on local sensitivity analysis and is strengthened via neural networks' function approximation abilities.

3.2 Gradient Ensemble Feature Selection

We now present our proposed methodology for feature selection using sparse neural networks, named **Grad**ient **En**semble **F**eature **S**election (**GradEnFS**). We start by describing our proposed neuron importance metric. Then, we explain the sparse MLP training process. We use the Sparse Evolutionary Training algorithm (Mocanu et al., 2018), the first DST algorithm, to train a sparse MLP. Finally, we describe how to perform feature selection at the end of training. Pseudocode for the GradEnFS algorithm can be found in Appendix A.

3.2.1 Neuron Importance Metric

Drawing inspiration from local sensitivity analysis, we measure how sensitive the loss is to changes in input neurons (features) within a neural network, quantifying this sensitivity as the importance of the neurons.

We use $\boldsymbol{s}(\boldsymbol{x}^{(i)})$ to represent the local sensitivity vector, which includes the local sensitivities of input neurons at a specific sample $\boldsymbol{x}^{(i)}$. $\boldsymbol{s}(\boldsymbol{x}^{(i)})$ is defined as follows:

$$\boldsymbol{s}(\boldsymbol{x}^{(i)}) = \nabla_{\boldsymbol{x}} \mathcal{L}(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}_s), y^{(i)}) = \nabla_{\boldsymbol{x}} \ell^{(i)} \tag{3}$$

where $y^{(i)}$ is the corresponding label of the sample $\boldsymbol{x}^{(i)}$, $\mathcal{L}$ is the loss function, $f(\boldsymbol{x}, \boldsymbol{\theta}_s)$ is a sparse neural network with parameter $\boldsymbol{\theta}_s$, and $\ell^{(i)}$ is the loss between the label and predicted label of the sample $\boldsymbol{x}^{(i)}$, and $\nabla_{\boldsymbol{x}} \ell^{(i)}$ denotes the gradient of loss with respect to feature vector $\boldsymbol{x}$.

To address bias resulting from measuring sensitivity using a single data sample, we calculate the overall sensitivity by averaging the absolute values of the local sensitivities obtained from a set of samples. Subsequently, we accumulate this overall sensitivity, often referred to as gradient information, from various models across different training iterations to achieve an ensemble effect as these models undergo updates in their topology and weight matrices over different iterations. We then consider the accumulated value as the neuron importance. We use $\boldsymbol{I}^{(t)}$ to represent the input neuron importance vector at iteration t. Initially, all elements of this vector are set to 0, and they are updated during the training iterations using the following formula:

$$\boldsymbol{I}^{(t)} = \beta \boldsymbol{I}^{(t-1)} + \frac{\sum_{i=0}^{N^{(t)}} |\boldsymbol{s}(\boldsymbol{x}^{(i)})|}{N^{(t)}} \tag{4}$$

where $N^{(t)}$ denotes the number of data samples in training iteration t, and β is the hyperparameter that determines the extent to which we account for historical effects.

In GradEnFS, we recommend constraining the value of β to $(0, 1]$. This recommendation is based on the principle that as the model approaches the underlying function of the real-world dataset, sensitivity analysis becomes more effective and accurate. Therefore, we should assign greater weight to the current results and less to historical results. Empirical evidence supporting this recommendation is presented in Appendix E. While the MLP is considered a universal approximator capable of approximating any real-world function with sufficient neurons, differences between the real function and the model still exist. Consequently, aggregating insights from various models becomes a valuable strategy. Thanks to the rapid convergence of MLP, the model often approximates the real-world distribution within just a few epochs. Additionally, the sparsity introduced to the model not only saves resources but also helps prevent overfitting during later training stages. Consequently, gathering sensitivity analysis information from models obtained in different training stages becomes beneficial.

Table 1: Datasets Characteristics

DATASET	TYPE	#FEATURES	#SAMPLES	#TRAIN	#TEST	#CLASSES
BASEHOCK	text	4862	1993	1595	398	2
PCMAC	text	3289	1943	1555	388	2
Prostate_GE	biological	5966	102	82	20	2
TOX_171	biological	5748	171	137	34	4
Madelon	artificial	500	2600	2080	520	2
Isolet	spoken letter recognition	617	1560	1248	312	26
COIL20	image	1024	1440	1152	288	20
USPS	hand written digit image	256	9298	7439	1859	10
MNIST	hand written digit image	784	70000	60000	10000	10

3.2.2 Sparse Multi-Layer Perceptron

Initialization GradEnFS initializes a sparse MLP as an Erdős-Rényi random graph, in which the sparse connections are uniformly distributed over the neurons from consecutive layers. The probability of establishing a connection between two neurons is determined by the following equation:

$$p(W_{ij}^k) = \frac{\epsilon(H^{k-1} + H^k)}{H^{k-1}H^k} \tag{5}$$

where ϵ is the hyperparameter to control the sparsity level, H^k denotes the number of neurons in k-th layer, and W_{ij}^k represents the connection between neuron i of layer $k-1$ and neuron j of layer k. Compared to completely random initialization or initialization with a fixed sparsity level in each layer, this probability distribution yields varying sparsity levels among different layers and promotes sparser connections between layers with more neurons.

Training The sparse MLP is trained to minimize the cross-entropy loss between the label y and the prediction $\hat{y}$. The loss function is as follows:

$$\mathcal{L}(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{6}$$

where C is the number of the classes, and $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta}_s)$ in which $f(\boldsymbol{x}; \boldsymbol{\theta}_s)$ is the sparse model with parameter $\boldsymbol{\theta}_s$. After calculating losses based on a batch of data, the model's sparse weight matrices are updated by **backpropagation** algorithm and utilize gradient information to **update the input neurons importance**. We simultaneously adapt the sparse topology through a **drop-and-grow** cycle after a fixed interval (batch or epoch).

Drop phase The algorithm first takes the absolute values of the weight of every existing connection. Following this, a fraction α of the existing connections is removed by selecting the smallest absolute value. This removal operation is performed layer by layer throughout the entire network.

Grow phase After the drop phase, the algorithm randomly regrows the same number of connections that were dropped layer by layer. This randomness offers several advantages. Firstly, it can potentially save computational resources since there is no need for computations like connection importance during regrowth (Evci et al., 2020; Atashgahi et al., 2022). Additionally, the randomness provides a vast exploration space for sparse topology, which enhances the ensemble effect.

3.2.3 Feature Selection

After the training process, we identify the K input neurons with the highest importance, considering them as the K most informative features.

4 EXPERIMENTS & RESULTS

4.1 Datasets

We evaluate the performance of our proposed method on nine publicly available datasets. The characteristics of these datasets are illustrated in Table 1.

4.2 Experimental Settings

Evaluation Metrics We evaluate the performance of feature selection algorithms from different aspects, including learning accuracy, memory usage, and computational cost. For **learning accuracy**, we assess it by classification accuracy, which means we train a classifier using the selected K features and obtain the test accuracy of this classifier as the evaluation. We use one of the most popular classifiers, Support Vector Machines (SVM) (Vapnik, 1992). Using the non-NN-based classifier ensures that our evaluation remains unbiased and avoids favoring NN-based baselines over others. Classification accuracy measured by other classifiers can be found in Appendix F.1. For **memory usage**, we calculate the number of network parameters used by each NN-based method. For **computational cost**, we assess the number of FLoating-Point Operations (FLOPs) needed for training a neural network. More details can be found in Appendix B.

Table 2: Classification Accuracy. The results are the SVM classification accuracy average over five independent runs (%). Bold fonts indicate the best performer and italic fonts indicate the second-best performer (Baseline is not under consideration).

METHOD	HIGH-DIMENSIONAL DATASETS				LOW-DIMENSIONAL DATASETS				
	BASEHOCK	PCMAC	Prostate_GE	TOX_171	Madelon	Isolet	COIL20	USPS	MNIST
Fisher Score	54.77 ± 0	57.22 ± 0	75.00 ± 0	73.53 ± 0	53.00 ± 0	77.88 ± 0	75.35 ± 0	93.60 ± 0	88.16 ± 0
CMIM	63.07 ± 0	61.86 ± 0	*90.00 ± 0*	70.53 ± 0	53.67 ± 0	82.37 ± 0	98.26 ± 0	95.05 ± 0	42.68 ± 0
ICAP	53.51 ± 0	62.37 ± 0	85.00 ± 0	64.71 ± 0	51.50 ± 0	77.24 ± 0	97.22 ± 0	95.60 ± 0	87.57 ± 0
RFS	67.34 ± 0	57.47 ± 0	**95.00 ± 0**	*79.41 ± 0*	54.00 ± 0	80.77 ± 0	87.50 ± 0	95.27 ± 0	40.60 ± 0
LassoNet	87.19 ± 0.84	*84.05 ± 0.83*	90.00 ± 1.23	73.53 ± 2.33	70.17 ± 0.89	87.82 ± 1.25	97.57 ± 0.72	95.32 ± 0.25	86.37 ± 1.01
STG	84.92 ± 1.04	81.44 ± 0.88	90.00 ± 2.01	**91.18 ± 2.53**	72.27 ± 1.24	80.77 ± 1.62	94.92 ± 0.64	96.18 ± 0.22	**94.09 ± 0.88**
QS	*90.15 ± 0.83*	83.75 ± 1.17	87.00 ± 6.78	65.88 ± 7.58	70.33 ± 0.43	*90.58 ± 2.13*	97.50 ± 0.97	96.14 ± 0.28	85.40 ± 2.07
GradEnFS_batch	**92.43 ± 1.29**	83.56 ± 0.75	89.00 ± 4.90	72.35 ± 4.78	*75.23 ± 2.50*	**92.69 ± 1.30**	**99.10 ± 0.47**	**97.22 ± 0.21**	80.94 ± 3.20
GradEnFS_epoch	88.94 ± 2.34	**84.38 ± 0.42**	84.00 ± 3.74	75.29 ± 3.98	**75.33 ± 3.28**	89.23 ± 0.77	*98.40 ± 1.47*	*96.65 ± 0.16*	*93.43 ± 0.51*
Baseline	96.73	88.92	90.00	82.35	59.33	93.59	98.96	97.20	97.82

Table 3: Memory Usage (#PARAMETERS $\times 10^5$) and Computational Cost (#FLOPS during training $\times 10^6$). The "SPARSE" columns include results for QS and GradEnFS, while the "DENSE_1" columns show the outcomes of LassoNet, and the "DENSE_3" columns show the outcomes of STG. The "DENSITY" column indicates the sparsity level of the sparse networks compared to their dense counterparts.

DATASETS	DENSITY	#PARAMETERS ($\times 10^5$)			#FLOPS ($\times 10^6$)		
		SPARSE	DENSE_1	DENSE_3	SPARSE	DENSE_1	DENSE_3
BASEHOCK	1.46%	1.00	48.64	68.64	0.60	29.18	41.18
PCMAC	1.61%	0.85	32.91	52.91	0.51	19.75	31.75
Prostate_GE	1.40%	1.12	59.68	79.68	0.67	35.81	47.81
TOX_171	1.43%	1.11	57.52	77.52	0.67	34.51	46.51
Madelon	0.23%	0.06	5.02	25.02	0.04	3.01	15.01
Isolet	2.30%	0.70	6.43	26.43	0.42	3.86	15.86
COIL20	2.50%	0.66	10.44	30.44	0.40	6.26	18.26
USPS	2.78%	0.63	2.66	22.66	0.38	1.60	13.60
MNIST	2.40%	0.67	7.86	27.94	0.41	4.72	16.76

Baselines We trained an SVM using the training dataset with all available features and measured the classification accuracy on the test dataset with all available features. This classification accuracy was considered as the baseline.

State-of-the-art methods We conducted a comparison of GradEnFS with seven state-of-the-art feature selection methods. These methods include filter methods such as Fisher score (Gu et al., 2012), CMIM (Fleuret, 2004), and CIAF (Jakulin, 2005), as well as embedding methods like RFS (Nie et al., 2010), QS (Atashgahi et al., 2020), STG (Yamada et al., 2020), and LassoNet (Lemhadri et al., 2021). Among them, STG and LassoNet are dense NN-based methods, and QS is sparse NN-based methods. More details about these methods can be found in Appendix C.

Implementation We implemented GradEnFS with PyTorch using the binary mask to simulate the sparsity. We also implemented QS with PyTorch and adapted it to work with a supervised MLP model. For filter methods and RFS, the implementations from the Scikit-Feature library[1] were utilized. For STG[2] and LassoNet[3], we employed the implementations provided by their respective authors.

Hyperparameter The architecture of the network used for GradEnFS, QS, and STG is a 3-hidden-layer MLP with 1000 neurons in each hidden layer. For LassoNet, we use a 1-hidden-layer MLP with 1000 neurons in the hidden layer, as running a 3-hidden-layer MLP would take an excessively long time (over 24 hours). The number of training epochs for all neural network methods is set to 100, with a learning rate of 0.001. For GradEnFS, the hyperparameters are set as follows: α is 0.3, β is 0.9, and ϵ is set to 10 for all datasets except for the Madelon dataset, where ϵ is set to 1. During the experiment, K values vary across 25, 50, 75, 100, 150, and 200 for all the datasets except for Madelon, for which K is fixed at 20, as this dataset contains only 20 informative features. Additional details on the hyperparameter settings for all experiments in this paper can be found in Appendix D. Moreover, we investigate the impact of hyperparameters on GradEnFS, and the results are presented in Appendix E.

[1] https://github.com/jundongl/scikit-feature

[2] https://github.com/runopti/stg

[3] https://github.com/lasso-net/lassonet

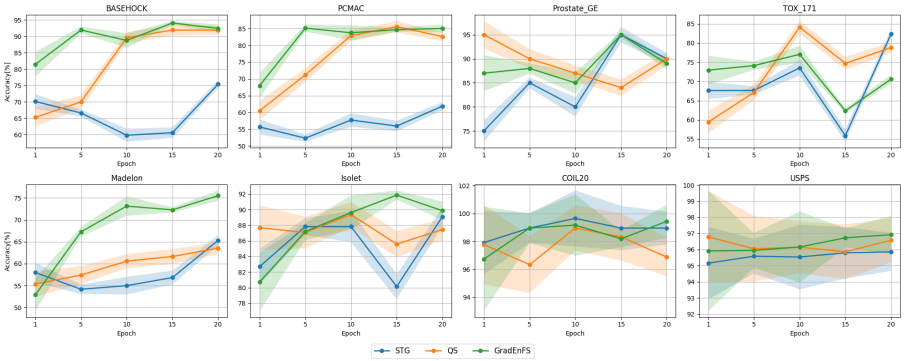

Figure 2: Classification Accuracy (%) of Different Methods After Different Epochs

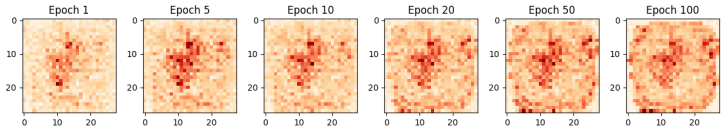

Figure 3: Heatmap Visualization of Neuron Importance of MNIST Dataset for Various Epochs.

4.3 Results

Classification Accuracy Table 2 shows the average classification accuracy from five independent runs for various feature selection methods. We conduct experiments with GradEnFS using two topology update intervals: after every batch (denoted as GradEnFS_batch) and after every epoch (denoted as GradEnFS_epoch). The results are based on 50 selected features for all the datasets except for Madelon, which used 20 selected features. Additional results for different values of K can be found in Appendix F.2.

For most of the datasets, the best performers are observed in NN-based methods, except for the Prostate_GE dataset, which demonstrates NN-based methods' superiority over non-NN-based methods. This distinction is particularly noticeable in two text datasets and the noisy Madelon dataset. We believe this is due to the complex dependencies between features within these three datasets.

GradEnFS performs decently on low-dimensional datasets, particularly on the Madelon dataset. The inherent sparsity in GradEnFS plays a crucial role in mitigating the adverse effects of noise features within the Madelon dataset, leading to improved and more accurate results for feature selection tasks. The performance of GradEnFS on biological datasets falls short of expectations, which could be attributed to the limited number of samples available. This method typically requires a decent number of samples to mitigate bias in neuron importance measurement. In a nutshell, GradEnFS performs best in six out of nine datasets.

Furthermore, we provide insight into the consistency between top features selected by our proposed method and various state-of-the-art feature selection methods in Appendix F.3, along with additional analysis in Appendices F.4 and F.5.

Memory Usage and Computational Cost Table 3 shows the memory usage (#PARAMETERS) and computational cost (#FLOPS) of each NN-based method. The results indicate that the sparse NN-based methods significantly reduce memory usage and computational costs by approximately 96% to 99% across various datasets compared to STG, which shares the same network architecture as the models used in the sparse NN-based methods in the experiment. Even in comparison to LassoNet, which uses only a single hidden layer MLP, memory usage and computational costs are reduced by 76% to 98%.

5 DISCUSSIONS

5.1 Converge Speed

We investigate the convergence behavior and speed of three NN-based feature selection methods: STG, QS, and GradEnFS. We conducted experiments over several epochs, ranging from 1 to 20, to observe how these methods perform. The results, illustrated in Figure 2, provide valuable insights into their behavior. More de-

tails about the convergence of the proposed algorithm can be found in Appendix G.

Across most datasets, GradEnFS exhibited a remarkable pattern. It shows a steep increase in accuracy from epoch 1 to 5 and achieves competitive accuracy by the 20th epoch, which is close to the results obtained after 100 epochs, as demonstrated in Table 2. This indicates that GradEnFS possesses a rapid convergence speed on the majority of datasets, with the Madelon dataset displaying this tendency prominently. Moreover, we observed that for most datasets, the standard deviation of GradEnFS's accuracy across the 20th epochs was considerably smaller than that observed at the 1st epoch. This suggests that the results became more stable as training progressed.

However, all methods exhibited substantial fluctuations in two biological datasets. This highlights the importance of having a sufficient number of samples when utilizing NN-based feature selection methods, as their performance is intricately linked to the training process of neural networks.

5.2 Visualization

Figure 3 offers a visualization of neuron importance across various epochs in the form of a heatmap. We perform this analysis on the MNIST dataset, an image dataset with handwritten digits centered in 28 $\times$ 28 grayscale images (see Appendix H). As the results show, GradEnFS swiftly identifies informative features, primarily located in the center of the image, even after just one epoch. Notably attractive is that, over subsequent epochs, GradEnFS gradually extends its focus beyond the central pixels, containing the neighboring pixels surrounding those composing the digit. This behavior suggests that GradEnFS begins to recognize the importance of pixels along the edges of the digit, implying that these edge pixels also contribute valuable information to the classification task. We also compared the visualization heatmaps between GradEnFS and QS; the details can be found in Appendix I.

5.3 Robustness in Noisy Dataset

Through our extensive experiments, we have observed that GradEnFS delivers significantly superior performance when compared to other methods on the challenging noisy dataset, Madelon. This dataset has 500 features, but only 20 of them are informative. To further investigate GradEnFS's robustness in handling noisy datasets, we have leveraged a function[4] provided by sklearn that enables the generation of artificial noisy datasets.

We generated multiple noisy datasets, each consisting of 2000 samples and 500 features, with varying numbers of informative features denoted as "K". These K informative features were intentionally placed at the first K indexes of the feature set. Consequently, we applied different feature selection methods to identify the indexes of the K informative features and compared them to the ground truth. The ratio of correctly identified informative features, referred to as the "hit rate", was calculated for each method.

The results are presented in Figure 4. From the results, it is evident that the NN-based method significantly outperforms the non-NN-based method. We believe this is due to the complex dependencies between the features in these artificial datasets. Neural networks can learn and model these intricate relationships. Among NN-based methods, we have observed that sparse NN-based methods clearly outperform dense-based methods in environments with high levels of noise (e.g., $K = 10$). This demonstrates that sparsity in neural networks plays a significant role in mitigating the noise effects in feature selection tasks. Moreover, GradEnFS emerges as the top performer across various K values in these artificial datasets, confirming the effectiveness of the sensitivity-based metric in this specific scenario. However, as the number of informative features increases, the hit rate of sparse NN-based methods decreases. We believe that when dealing with datasets that contain a greater number of informative features, it may need to employ a denser network with more parameters for effective analysis. However, throughout our experiments, we maintained a consistently high sparsity level ($\epsilon = 1$). Therefore, increasing the network's density slightly for larger values of K could potentially improve the hit rate.

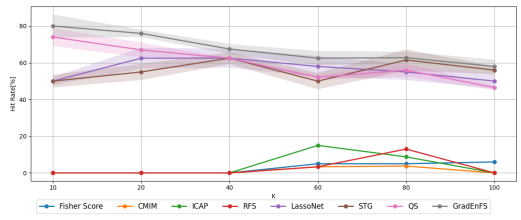

Figure 4: Hit Rate (%) of Various Methods on Noisy Datasets with Varying Numbers of Informative Features (K).

6 CONCLUSIONS

In this paper, we proposed an efficient sparse NN-based feature selection method named GradEnFS.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

This method trains a sparse MLP from scratch and, during the training, measures the importance of the input neurons using the sensitivity-based metric. Finally, at the end of the training, it selects the top-K features based on their importance. We performed extensive experiments to compare our proposed method with other state-of-the-art methods on nine datasets. GradEnFS achieves the best performance on six datasets regarding feature selection accuracy and has obvious superiority over other NN-based methods in terms of resource consumption. Furthermore, we demonstrated that GradEnFS excels in handling noisy datasets compared to other methods.

Limitations Due to the absence of hardware capable of efficiently supporting sparse matrix operations, the sparse neural network in GradEnFS is currently implemented using masks to simulate sparsity, which is a common approach in most related works. Consequently, the potential advantages of sparse networks in terms of resource consumption have not been fully harnessed. However, there is a recent and increasing focus on developing hardware and software support for the sparsity of neural networks (Liu et al., 2021; Curci et al., 2021). This advancement holds the potential for a fully sparse implementation of the proposed method, although it falls beyond the scope of our current study.

References

A. Abid, M. F. Balin, and J. Zou. Concrete autoencoders for differentiable feature selection and reconstruction. *arXiv preprint arXiv:1901.09346*, 2019.

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. *arXiv preprint arXiv:2012.00560*, 2020.

Z. Atashgahi, J. Pieterse, S. Liu, D. C. Mocanu, R. Veldhuis, and M. Pechenizkiy. A brain-inspired algorithm for training highly sparse neural networks. *Machine Learning*, 111(12):4411–4452, 2022.

S. Bibikar, H. Vikalo, Z. Wang, and X. Chen. Federated dynamic sparse training: Computing less, communicating less, yet learning better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6080–6088, 2022.

G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

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

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

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

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

S. Ding. Feature selection based f-score and aco algorithm in support vector machine. In *2009 Second International Symposium on Knowledge Acquisition and Modeling*, volume 1, pages 19–23. IEEE, 2009.

N. El Aboudi and L. Benhlima. Review on wrapper feature selection approaches. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–5, 2016. doi: 10.1109/ICEMIS.2016.7745366.

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*, pages 2943–2952. PMLR, 2020.

F. Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine learning research*, 5(9), 2004.

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

Q. Gu, Z. Li, and J. Han. Generalized fisher score for feature selection. *arXiv preprint arXiv:1202.3725*, 2012.

K. Han, Y. Wang, C. Zhang, C. Li, and C. Xu. Autoencoder inspired unsupervised feature selection. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 2941–2945. IEEE, 2018.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

A. Jakulin. *Machine learning based on attribute interactions*. PhD thesis, Univerza v Ljubljani, 2005.

S. A. Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600, 1989.

Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

N. Lee, T. Ajanthan, and P. H. Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

S. Lee, Y.-T. Park, B. J. d'Auriol, et al. A novel feature selection method based on normalized mutual information. *Applied Intelligence*, 37(1):100–120, 2012.

I. Lemhadri, F. Ruan, and R. Tibshirani. Lassonet: Neural networks with feature sparsity. In *International Conference on Artificial Intelligence and Statistics*, pages 10–18. PMLR, 2021.

J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6): 94, 2018.

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*, 33(7):2589–2604, 2021.

C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

Y. Lu, Y. Fan, J. Lv, and W. Stafford Noble. Deeppink: reproducible feature selection in deep neural networks. *Advances in neural information processing systems*, 31, 2018.

S. Maldonado and R. Weber. A wrapper method for feature selection using support vector machines. *Information Sciences*, 179(13):2208–2217, 2009. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2009.02.014. URL `https://www.sciencedirect.com/science/article/pii/S0020025509000917`. Special Section on High Order Fuzzy Sets.

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*, 9(1):1–12, 2018.

D. C. Mocanu, E. Mocanu, T. Pinto, S. Curci, P. H. Nguyen, M. Gibescu, D. Ernst, and Z. A. Vale. Sparse training theory for scalable and efficient agents. *arXiv preprint arXiv:2103.01636*, 2021.

D. C. Mocanu et al. *Network computations in artificial intelligence*. Technische Universiteit Eindhoven, 2017.

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

P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

M. C. Mozer and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

F. Nie, H. Huang, X. Cai, and C. Ding. Efficient and robust feature selection via joint 2, 1-norms minimization. *Advances in neural information processing systems*, 23, 2010.

J. Pizarroso, J. Portela, and A. Muñoz. Neuralsens: sensitivity analysis of neural networks. *arXiv preprint arXiv:2002.11423*, 2020.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

D. Singh, H. Climente-González, M. Petrovich, E. Kawakami, and M. Yamada. Fsnet: Feature selection network on high-dimensional biological data. *arXiv preprint arXiv:2001.08322*, 2020.

D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

G. Sokar, D. C. Mocanu, and M. Pechenizkiy. Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021a.

G. Sokar, E. Mocanu, D. C. Mocanu, M. Pechenizkiy, and P. Stone. Dynamic sparse training for deep reinforcement learning. *arXiv preprint arXiv:2106.04217*, 2021b.

G. Sokar, Z. Atashgahi, M. Pechenizkiy, and D. C. Mocanu. Where to pay attention in sparse training for feature selection? In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=xWvI9z37Xd`.

S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 138–145, 2017.

N. Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth annual conference of the international speech communication association*, 2015.

H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neu-*

ral Information Processing Systems, 33:6377–6389, 2020.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT '92)*, 1992. URL https://link.springer.com/chapter/10.1007/3-540-57256-3_5.

C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Y. Yamada, O. Lindenbaum, S. Negahban, and Y. Kluger. Feature selection using stochastic gates. In *International Conference on Machine Learning*, pages 10648–10659. PMLR, 2020.

X. Zhou and H. Lin. *Local Sensitivity Analysis*, pages 616–616. Springer US, Boston, MA, 2008. ISBN 978-0-387-35973-1. doi: 10.1007/978-0-387-35973-1_703. URL https://doi.org/10.1007/978-0-387-35973-1_703.

H. Zhu and Y. Jin. Multi-objective evolutionary federated learning. *IEEE transactions on neural networks and learning systems*, 31(4):1310–1322, 2019.

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

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Not Applicable
   (b) Complete proofs of all theoretical results. Not Applicable
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Not Applicable
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

A Pseudocode of GradEnFS

Algorithm 1: Gradient Ensemble Feature Selection Algorithm

Data: the data matrix $\boldsymbol{D}(\boldsymbol{X}, \boldsymbol{y})$
Result: indices of selected K features stored in $\mathbb{F}_s$
set ϵ, α, β, K;
initialize an MLP with a specified topology based on the formula $p(W_{ij}^k) = \frac{\epsilon(H^{k-1}+H^k)}{H^{k-1}H^k}$;
initialize dynamic input neuron importance vector $\boldsymbol{I}$ with every element set to 0;
for *each training iteration* **do**
  perform feedforward and backpropagation;
  compute the local sensitivity for every input neuron on every sample according to formula $\boldsymbol{s}(\boldsymbol{x}^{(i)}) = \nabla_{\boldsymbol{x}}\mathcal{L}(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}_s), y^{(i)}) = \nabla_{\boldsymbol{x}}\ell^{(i)}$;
  update neuron importance vector $\boldsymbol{I}$ according to formula $\boldsymbol{I}^{(t)} = \beta\boldsymbol{I}^{(t-1)} + \frac{\sum_{i=0}^{N^{(t)}} |\boldsymbol{s}(\boldsymbol{x}^{(i)})|}{N^{(t)}}$;
  perform weights update;
  if *is the topology update interval* **then**
    for *each* $\boldsymbol{W}_s^k$ *of MLP* **do**
      drop $\alpha\|\boldsymbol{W}_s^k\|_0$ connections that are closest to 0 from $\boldsymbol{W}_s^k$;
      if *is not the last topology update interval* **then**
        regrow $\alpha\|\boldsymbol{W}_s^k\|_0$ connections to $\boldsymbol{W}_s^k$ randomly;
      end
    end
  end
end
$\mathbb{F}_s \leftarrow top(\boldsymbol{I}, K)$;

B Details of Evaluation Metrics

In our experiments, we measure the resource comsuption of the neural network based methods by the number of parameters and the number of FLoating-Point Operations (FLOPs). Since existing deep learning hardware isn't optimized for sparse matrix computations, most methods for obtaining sparse neural networks simulate sparsity using binary masks over the weights, including our proposed method. Consequently, the runtimes of these methods aren't reliable indicators of their efficiency. Instead, we turn to FLOPs, a measure of the number of floating-point operations required for a computation. In the context of deep learning models, FLOPs are commonly used to assess the efficiency of a sparse neural network compared to its dense counterpart. To calculate the FLOPs needed for a single computation (including the forward pass, backpropagation, and parameters update) of the model, we count the total number of multiplications and additions layer by layer. The overall FLOPs is then derived by summing these operations. Furthermore, we compare the parameter count of all neural network-based methods, as the number of parameters indicates the model's size, which directly impacts memory consumption and computational complexity.

Here, we provide a step-by-step guide on calculating the number of FLOPs for an example using the MNIST dataset. We employ a dense MLP with three hidden layers for training on the MNIST dataset. The network structure is defined as (784, 1000, 1000, 1000, 10), where each number represents the number of neurons in each respective layer.

When using a sample for one complete training, the process involves three main steps: the forward pass, backpropagation, and parameter updates. Therefore, we break down the FLOPs calculation into these three distinct parts.

- Forward Pass: Calculate the weighted sum of inputs and apply the activation function for each layer. Each neuron in each layer performs two floating-point operations (one multiplication and one summation).
- Backpropagation: Calculate gradients for each layer, which involves the same number of FLOPs as the

forward pass.

- Parameter Updates: For each weight parameter in the network, update it using the gradients and the learning rate (one multiplication and one summation for every weight).

Now, we start to calculate for the mnist example:

- Forward Pass:
  - Input Layer to Hidden Layer 1: 784 (input features) * 1000 (neurons) * 2 = 1,568,000 FLOPs
  - Hidden Layer 1 to Hidden Layer 2: 1000 * 1000 * 2 = 2,000,000 FLOPs
  - Hidden Layer 2 to Hidden Layer 3: 1000 * 1000 * 2 = 2,000,000 FLOPs
  - Hidden Layer 3 to Output Layer: 1000 * 10 * 2 = 20,000 FLOPs
- Backpropagation: Total number of FLOPs is the same as the forward pass.
- Parameter Updates: Total number of FLOPs is two times of the number of parameters.
  - Input Layer to Hidden Layer 1: 784 (input features) * 1000 (neurons) * 2 = 1,568,000 FLOPs
  - Hidden Layer 1 to Hidden Layer 2: 1000 * 1000 * 2 = 2,000,000 FLOPs
  - Hidden Layer 2 to Hidden Layer 3: 1000 * 1000 * 2 = 2,000,000 FLOPs
  - Hidden Layer 3 to Output Layer: 1000 * 10 * 2 = 20,000 FLOPs

Then, we can sum up and get the total FLOPs for one complete training in this MLP architecture, the number are approximately 16,764,000 FLOPs. Please note that this is a simplified estimate, and actual FLOP counts can vary depending on factors like batch size and hardware optimizations.

C Details of Other State-of-the-art Methods

In our experiments, we employed seven widely recognized feature selection methods, comprising three filter methods and four embedding methods. Here is a concise description of each of them.

The three filter methods are:

- Fisher Score: selects features that maximize the similarity of feature values among the same class.
- ICAP: a methodology for feature selection based on information theory principles. The approach focuses on identifying and utilizing attribute interactions and employs probability theory, entropy, and Kullback-Leibler divergence as loss functions, with a Bayesian framework for significance testing and uncertainty handling.
- CMIM: a feature selection technique based on conditional mutual information, which iteratively selects features that maximize the mutual information with the class labels given the selected features.

The four embedding methods are:

- LassoNet: utilizes a neural network with residual connections to the input layer and solves a two-component (linear and non-linear) optimization problem to determine feature importance.
- STG: utilizes a continuous relaxation of the Bernoulli distribution in a neural network to perform feature selection.
- Quick Selection (QS): selects features using the magnitude-based neuron importance metric in a sparse neural network.
- Robust Feature Selection (RFS): employs joint l2,1-norm minimization on the loss function and regularization to select features.

D Additional Details of Hyperparameters Setting

In this section, we will describe the hyperparameters setting for different experiments. All experiments were conducted on the CentOS 7.9 operating system and executed on a CPU. We conducted zero-mean-unit variance normalization before applying various feature selection methods to these nine datasets. This normalization step was essential not only for LassoNet but also beneficial for our proposed method. Then, we split the datasets, except for MNIST, which is already separated by the provider, by 80% and 20% as the training and testing sets, respectively. We further separated the training sets into 90% and 10%, of which 90% of data was used for training, and the other 10% was used as a validation set to monitor if the hyperparameters were proper for the model training. Notably, for the experiment on the MNIST dataset, we used part of the datasets to train GradEnFS with batch update. This decision was made due to the considerable time required when using the entire dataset for five runs in GradEnFS with batch updates, which would exceed 24 hours. Since we set a time constraint of under 24 hours for all experiments, GradEnFS with batch updates was run with a reduced dataset, utilizing 6,000 training samples and 1,000 testing samples for the MNIST dataset. We use the whole training and testing datasets (with 60,000 training samples and 10,000 testing samples) for all the other methods, including GradEnFS with epoch updates.

D.1 Experiment on classification accuracy

To ensure a fair comparison among various feature selection algorithms, we chose an appropriate value for the hyperparameters shared across multiple algorithms. For hyperparameters unique to each algorithm, we employed a grid search to find a proper value among a small set of values.

Therefore, the network used in the experiments for all neural-network-based feature selection methods, except for LassoNet, is an MLP with three hidden layers, and each hidden layer has 1000 neurons. The activation function used for the hidden layers is ReLU, and the activation function of the output layer is Softmax. The optimizer of the network is Adam. Given that our proposed method relies on gradient flow information, for GradEnFS, we incorporate batch normalization after each activation function to optimize the gradient flow, thus improving the performance of our method. For LassoNet, we chose a neural network with a single hidden layer of 1000 neurons. This choice was made due to practical considerations. A neural network with three hidden layers not only required extensive time to complete the experiments (over 24 hours), but it also yielded SVM accuracy results that were inferior to those achieved with the single hidden layer network for some datasets. Consequently, we opted to report the results based on a single hidden layer model in our experiments.

The hyperparameters for training neural networks, including batch size, learning rate, and the number of epochs, have been set to 100 (except for the two biological datasets, which batch size has been set to 50), 0.001, and 100, respectively. As the number of informative features is usually unknown for most datasets, we employed multiple K values to obtain diverse results when applying each method on datasets except for the Madelon dataset, where K represents the number of features in the final selected informative feature subset. The chosen K values include 25, 50, 75, 100, 150, and 200. For the Madelon dataset, K was set to 20, as this dataset contains only 20 informative features.

For GradEnFS, the rewire rate α and the historical neuron importance ratio β were set to specific values of 0.3 and 0.9, respectively. The grid search range for α contained [0.1, 0.3, 0.5, 0.7, 0.9], and β contained [0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0]. The sparsity level ϵ is adjusted based on the dataset. Specifically, ϵ is set to 1 for the Madelon dataset. For the remaining datasets, ϵ is set to 10. The grid search range for ϵ contained [1, 10, 20, 30, 50, 70, 100, 150, 200, ∞]. Notably, when ϵ was set to ∞, the network transformed into a dense neural network.

For QS, we maintained the same values for ϵ and α as in GradEnFS according to various datasets. In the case of RFS, the hyperparameter γ was set to 1, with a search range of [0.1, 0.5, 1, 10]. The hyperparameter λ for STG was set to 0.5, with a search range of [0.1, 0.5, 1, 10], while the hyperparameter M for LassoNet was set to 10, as recommended by the authors who proposed this algorithm.

D.2 Experiment on converge speed

In this experiment, our objective is to compare the convergence speed of three feature selection methods, QS, STG, and GradEnFS (with batch updates). Notably, we chose not to include Lassonet in this comparison since it utilizes a different network structure in our experiment and differs significantly from the three methods under

investigation. As a result, we opted to omit Lassonet to ensure fair competition among the selected methods. Our choice to focus on GradEnFS with batch updates is based on the observation that most datasets demonstrated a better performance when batch updates were employed in the results of classification accuracy. In cases where epoch updates are more suitable, the performance of the two topology update intervals is similar, with one notable exception: the MNIST dataset, where epoch updates outperform batch updates and demonstrate significantly faster training. Finally, the remaining hyperparameters and settings for this experiment are consistent with those described in Section D.1.

D.3 Experiment on visualization

In this experiment, we have adopted GradEnFS with epoch update intervals as our chosen method. This choice is influenced by the advantages in terms of classification accuracy and training speed that epoch updates exhibit over batch updates in training on the MNIST dataset. Other selected hyperparameters for GradEnFS include $\epsilon = 50$, $\alpha = 0.3$, and $\beta = 0.9$. For the QS method, we maintain consistency by using the same hyperparameters $\epsilon = 50$ and $\alpha = 0.3$. Furthermore, the network structure remains unchanged as described in Section D.1.

D.4 Experiment on robustness in noisy datasets

In this experiment, we employ GradEnFS with epoch update intervals, where we set $\epsilon = 1$, $\alpha = 0.3$, and $\beta = 0.9$ as the chosen hyperparameters. For all other methods used in this experiment, the hyperparameter settings remain consistent with those described in Section D.1.

Regarding data generation, we configure the following parameters: 2000 samples, 500 features, 2 classes, 20 informative features, a random state of 0, and shuffle set to false. All other hyperparameters for data generation are maintained at their default values.

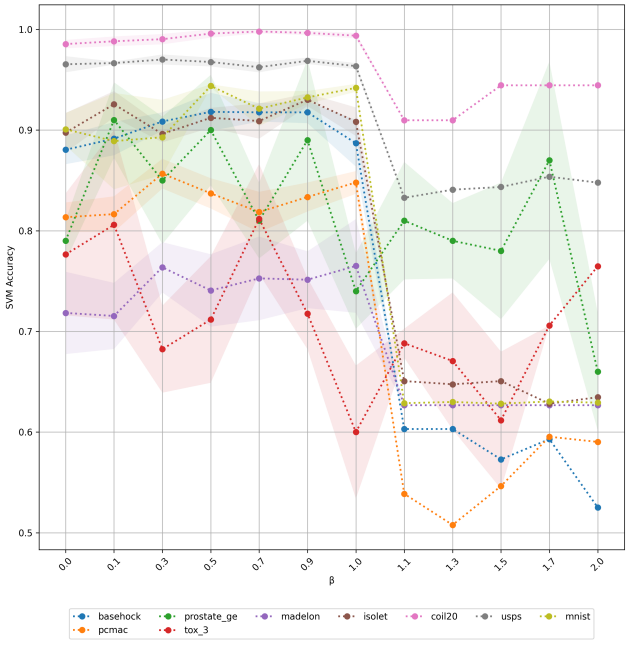

Figure 5: Classification Accuracy of GradEnFS with Different Betas

E Hyperparameters Effect

In this section, we delve into the analysis of the impact of hyperparameters on the quality of GradEnFS in feature selection tasks. The hyperparameters include ϵ, which controls the sparsity level, the rewire rate α, the parameter controlling historical neuron importance ratio β and network's width w.

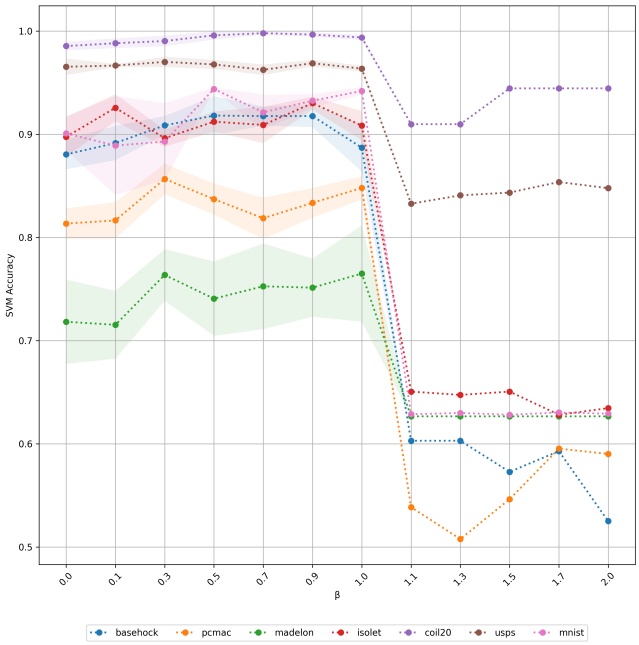

Figure 6: Classification Accuracy of GradEnFS with Different β Values (Excluding Biological Datasets)

The experimental settings were as follows: for the parameter ϵ, we varied its values within the range [1, 10, 20, 30, 50, 70, 100, 150, 200, ∞]. When $\epsilon = \infty$, the network is a dense neural network. Throughout this experiment, the parameter α remained constant at 0.3, and β was set to 0.9. In the α experiment, we explored values in the range [0.1, 0.3, 0.5, 0.7, 0.9], with β held at 0.9 and ϵ set to 10 for all datasets except for Madelon, where ϵ was set to 1. In the β experiment, we considered values within the range [0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.1, 1.3, 1.5, 1.7, 2.0], while α remained at 0.3, and ϵ was set to 10 for all datasets except for Madelon, where ϵ was set to 1. We use a batch update for all the datasets except the MNIST dataset, which is an epoch update. We consistently retained K = 50 features throughout all experiments, except for Madelon, where K was set to 20. We employed SVM to measure the accuracy of the selected features. In the network width w experiment, we explored widths of 500 and 1500 with sparsity ranging from 95% to 99%. Throughout this experiment, the parameter α remained constant at 0.3, while β was set to 0.9.

Beta The results for β are presented in Figure 5. However, as previously discussed, given that the number of samples in biological datasets is relatively small, the results for these datasets exhibit significant fluctuations. For better analysis, we provide a separate Figure 6 that excludes the results of biological datasets.

When β was set to 0, the algorithm computed neuron importance solely based on the final trained model without considering any historical or ensemble effects, resulting in a loss of the benefits associated with ensemble learning. This phenomenon is clearly observable in Figure 6 where the majority of datasets (with the exception of USPS) exhibit an increase in accuracy as β increases within the range [0,1]. Even for USPS, where the accuracies exhibit minimal variation as β changes within the range [0,1], the ensemble effect still brings an advantage, as evidenced by the reduction in standard deviation as β increases within this range.

When β exceeded a value of 1, a significant drop in performance is evident. This occurs because the algorithm assigns the highest weight to the initial untrained sparse model but the smallest weight to the well-trained final models, damaging performance. Hence, we recommend constraining this hyperparameter within the range (0,1] for optimal performance.

Epsilon and Alpha From Figure 7, it is evident that most of the datasets achieve decent performance when ϵ is set to 10, with the exception of the Madelon and biological datasets. The Madelon dataset, given its abundance of noisy features, benefits from a smaller ϵ to make the network sparser and mitigate the impact of noise. The

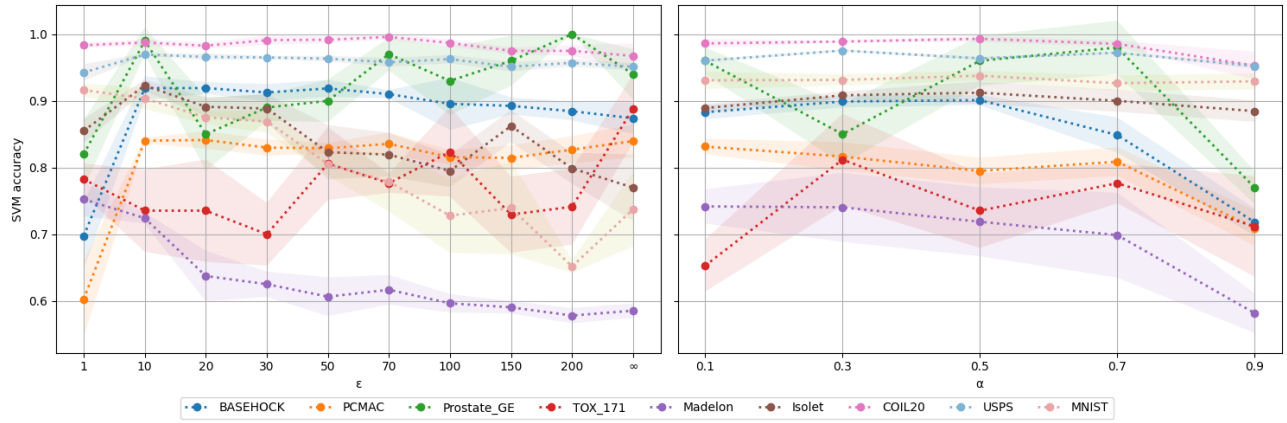

Figure 7: Classification Accuracy of GradEnFS with Different ϵ or α Values

performance of the biological dataset remains unstable due to limited sample size, yet there is a discernible trend of increased accuracy as ϵ increases, which may suggest that, for such high-dimensional and intricate datasets, a denser network is preferable for improved analysis. Most datasets are not particularly sensitive to changes in α. However, considering that some datasets experience a decline in performance at high α values, it is advisable to set this hyperparameter to a moderate value for optimal results.

Width The results for w are presented in Table 4. The results show that PCMAC and MNIST had slight performance drops at width=500 compared to higher widths. Attempts to lower the sparsity to 85%, 60%, and 45% (which are not shown in the table) resulted in worsened performance due to increased noise. We inferred that these datasets benefit from increased network complexity with increased width rather than connections (density). Thus, a high sparsity range of 95% to 98% yields optimal performance across various widths. Besides, width=1500 didn't notably improve over width=1000 (see Table 2). Hence, we chose width=1000 across all datasets for a good performance efficiency trade-off.

Table 4: Exploration on width selection. The results are the SVM classification accuracy average over five independent runs (%).

SETTING			DATASETS								
width	epsilon	approximate sparsity	BASEHOCK	PCMAC	Prostate_GE	TOX_171	Madelon	Isolet	COIL20	USPS	MNIST
500	1	99%	84.07 ± 1.94	78.76 ± 3.26	88.00 ± 9.27	76.47 ± 6.17	77.20 ± 2.25	88.14 ± 2.50	99.24 ± 0.55	96.99 ± 0.19	91.36 ± 0.43
	5	98%	87.79 ± 1.10	81.24 ± 2.40	79.00 ± 5.83	66.47 ± 5.13	76.00 ± 3.39	88.01 ± 2.21	99.38 ± 0.26	96.84 ± 0.14	92.44 ± 1.42
	10	95%	89.00 ± 0.69	82.68 ± 1.11	92.00 ± 8.12	78.82 ± 6.81	69.77 ± 4.39	89.29 ± 0.56	99.31 ± 0.49	96.77 ± 0.19	91.60 ± 1.75
1500	1	99%	83.42 ± 1.80	80.10 ± 0.91	80.00 ± 8.94	75.29 ± 5.46	68.20 ± 0.91	91.86 ± 1.44	99.58 ± 0.34	96.36 ± 0.29	94.33 ± 0.85
	10	98%	88.94 ± 0.78	86.03 ± 1.34	91.00 ± 3.74	85.88 ± 7.30	66.40 ± 0.98	88.14 ± 2.10	98.06 ± 0.52	96.35 ± 0.41	93.73 ± 0.75
	30	95%	91.71 ± 0.94	85.41 ± 1.61	99.00 ± 2.00	73.53 ± 4.92	61.50 ± 1.70	79.87 ± 4.56	98.61 ± 0.69	96.25 ± 0.16	93.13 ± 0.45

F Additional Results of Classification Accuracy

F.1 Classification accuracy of KNN, ExtraTrees and Sparse MLP trained by SET

The classification accuracies evaluated by KNN and ExtraTree are listed in Table 5 and Table 6, respectively. We also present the results of using SET to train a sparse MLP on the feature subsets selected by GradEnFS in Table 7. For most datasets, these results were similar to those of SVM, meaning our selected subsets are robust to various classifiers.

Table 5: Classification Accuracy. The results are the KNN classification accuracy average over five independent runs (%). Bold fonts indicate the best performer and italic fonts indicate the second-best performer (Baseline is not under consideration).

METHOD	HIGH-DIMENSIONAL DATASETS				LOW-DIMENSIONAL DATASETS				
	BASEHOCK	PCMAC	Prostate_GE	TOX_171	Madelon	Isolet	COIL20	USPS	MNIST
Fisher Score	57.54 ± 0	58.25 ± 0	80.00 ± 0	64.71 ± 0	52.50 ± 0	69.55± 0	77.78± 0	91.07± 0	82.91± 0
CMIM	54.52 ± 0	53.09 ± 0	75.00 ± 0	64.71 ± 0	49.50 ± 0	68.59 ± 0	96.53± 0	94.67± 0	36.52 ± 0
ICAP	56.28 ± 0	55.93 ± 0	70.00 ± 0	67.65 ± 0	50.33 ± 0	68.59 ± 0	91.32± 0	94.67± 0	85.12± 0
RFS	57.04 ± 0	60.82 ± 0	75.00 ± 0	44.12 ± 0	54.33 ± 0	71.47 ± 0	87.50± 0	94.46± 0	53.64± 0
LassoNet	87.44 ± 2.35	*83.25 ± 3.68*	*90.00 ± 5.23*	**79.41 ± 7.67**	61.83 ± 4.32	69.55 ± 2.31	95.14 ± 0.89	94.57 ± 1.11	82.07 ± 3.37
STG	83.17 ± 2.11	82.99 ± 1.37	85.00 ± 5.96	73.53 ± 5.32	71.67 ± 4.67	74.36 ± 1.66	*97.22 ± 1.12*	95.37 ± 0.89	**90.73 ± 2.51**
QS	*88.39 ± 2.20*	**86.60 ± 2.06**	73.00 ± 9.27	74.71 ± 4.40	68.17 ± 4.31	81.41 ± 2.17	93.12 ± 1.69	**96.09 ± 0.60**	85.53 ± 1.74
GradEnFS_batch	**88.74 ± 1.97**	79.23 ± 1.86	**99.00 ± 2.00**	*76.47 ± 7.67*	**74.93 ± 4.68**	**86.54 ± 1.48**	**98.75 ± 0.35**	95.93 ± 0.34	74.70 ± 3.79
GradEnFS_epoch	86.23 ± 1.57	81.70 ± 2.35	74.00 ± 5.83	74.71 ± 5.46	*73.30 ± 5.59*	*81.99 ± 1.52*	93.75 ± 1.05	*96.04 ± 0.50*	*87.84 ± 2.22*
Baseline	78.89	73.97	75.00	79.41	47.67	80.13	99.31	96.29	92.10

Table 6: Classification Accuracy. The results are the ExtraTree classification accuracy average over five independent runs (%). Bold fonts indicate the best performer and italic fonts indicate the second-best performer (Baseline is not under consideration).

METHOD	HIGH-DIMENSIONAL DATASETS				LOW-DIMENSIONAL DATASETS				
	BASEHOCK	PCMAC	Prostate_GE	TOX_171	Madelon	Isolet	COIL20	USPS	MNIST
Fisher Score	60.30 ± 0	61.08 ± 0	85.00 ± 0	67.65 ± 0	52.17 ± 0	78.53 ± 0	85.76 ± 0	93.49 ± 0	86.77 ± 0
CMIM	64.57 ± 0	53.61 ± 0	**95.00 ± 0**	**82.35 ± 0**	53.83 ± 0	86.54 ± 0	97.92 ± 0	95.00 ± 0	43.71 ± 0
ICAP	56.03 ± 0	56.96 ± 0	65.00 ± 0	**82.35 ± 0**	50.50 ± 0	85.90 ± 0	98.96 ± 0	95.64 ± 0	87.97 ± 0
RFS	64.07 ± 0	61.08 ± 0	**95.00 ± 0**	73.53 ± 0	60.33 ± 0	*90.06 ± 0*	92.71 ± 0	95.56 ± 0	60.17 ± 0
LassoNet	91.46 ± 1.58	86.86 ± 2.13	95.00 ± 3.69	76.47 ± 10.26	67.17 ± 5.24	83.65 ± 1.58	*99.31 ± 0.98*	95.43 ± 0.68	86.33 ± 3.89
STG	89.95 ± 1.14	87.89 ± 1.55	85.00 ± 2.17	88.24 ± 10.31	82.17 ± 4.69	82.69 ± 1.38	99.00 ± 1.11	*95.80 ± 0.32*	**92.85 ± 3.52**
QS	*94.23 ± 1.36*	*87.89 ± 0.63*	78.00 ± 6.78	68.82 ± 8.24	77.73 ± 7.38	84.49 ± 0.77	99.10 ± 0.64	94.67 ± 0.52	89.03 ±1.27
GradEnFS_batch	93.07 ± 0.80	**88.20 ± 0.85**	91.00 ± 2.00	74.12 ± 6.28	**86.53 ± 2.65**	**92.31 ± 1.25**	92.31 ± 1.25	**95.88 ± 0.34**	80.29 ± 3.83
GradEnFS_epoch	**94.37 ± 1.20**	86.34 ± 0.92	92.00 ± 4.00	71.76 ± 10.29	*83.73 ± 4.19*	86.28 ± 1.81	**99.86 ±0.28**	94.78 ± 0.33	*90.85 ± 1.61*
Baseline	98.74	94.07	90.00	67.65	69.00	93.59	99.65	97.04	95.30

F.2 SVM classification accuracy of different values of K

We present the detailed results of accuracies on various values of K in Figure 8. Our experimental outcomes are derived from averaging results obtained from five individual runs, ensuring the reduction of any potential bias.

According to the result, a clear trend emerges in two text datasets (BASEHOCK and PCMAC): neural network-based feature selection methods tend to outperform classical feature selection approaches across various datasets, proving the advantage of utilizing neural networks' ability to capture non-linear relationships between features to perform feature selection. We also noted that across all datasets, as the value of K increases, the SVM accuracy of GradEnFS either increases or remains constant. This consistent trend suggests that whenever GradEnFS adds a feature to the informative subset, it adds a helpful feature or, at the very least, is not noisy. This contrasts with other methods where SVM accuracy often fluctuates when increasing K (especially in two biological datasets (Prostate_GE and TOX_171), indicating that these methods occasionally introduce noisy features into the selection.

F.3 Consistency with state-of-the-art feature selection methods

We believe methods with the highest SVM accuracy exhibit higher consistency in selected features. For analysis, we utilized a heatmap to visualize the percentage of shared features within the selected subset among various methods on the Madelon dataset in Figure 9. We selected Madelon, which has 20 informative features among 500 total features, because in this dataset, neural network-based methods, including our method GradEnFS, notably achieve higher SVM accuracies compared to classical methods (refer to Table 2 in the paper). The plot underscores the consistency among the feature subsets selected by GradEnFS and other high-performing methods (STG, LassoNet, and QS, all with accuracies greater than 70%) while differing from methods with poor performance (RFS, ICAP, CMIM, and Fisher Score, all with accuracies less then 55%).

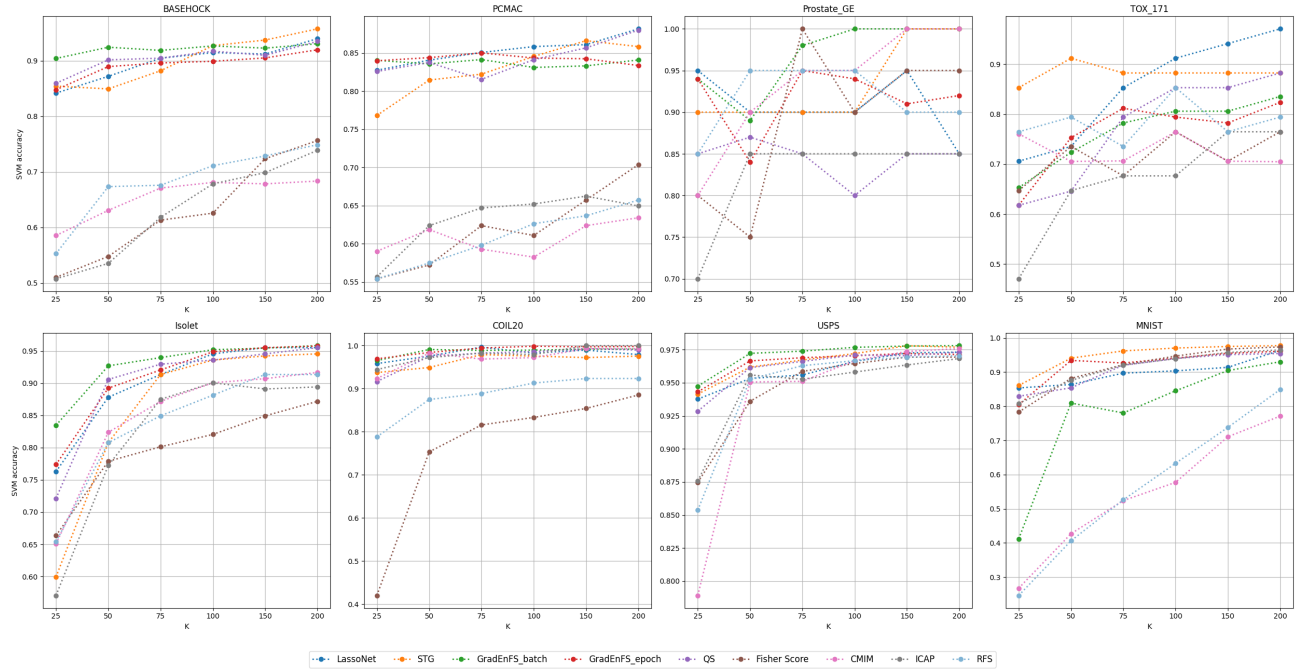

Figure 8: Feature selection comparison results for all datasets, including accuracy for various values of K.

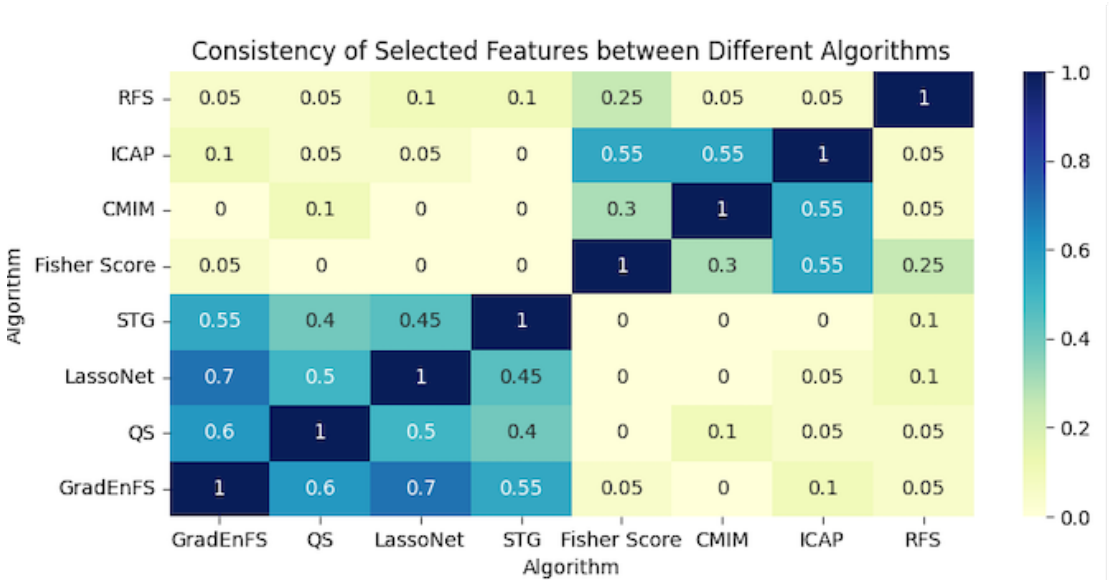

Figure 9: Heatmap about the percentage of shared features within the selected subset among various methods on the Madelon dataset.

Table 7: Classification Accuracy. The results are the classification accuracies evaluated by sparse MLPs with various sparsities and average over five independent runs (%).

SETTING		DATASETS								
Method	Epsilon	BASEHOCK	PCMAC	Prostate_GE	TOX_171	Madelon	Isolet	COIL20	USPS	MNIST
SET	1	89.10 ± 5.22	73.77 ± 12.09	92.00 ± 6.71	73.53 ± 14.26	83.23 ± 1.11	89.10 ± 2.28	99.10 ± 0.40	95.68 ± 0.66	85.86 ± 2.09
	30	92.11 ± 1.67	83.45 ± 4.15	92.00 ± 5.70	82.94 ± 8.92	81.43 ± 2.14	92.44 ± 1.92	99.17 ± 0.31	96.92 ± 0.38	88.68 ± 2.46
	100	91.66 ± 1.75	83.30 ± 3.14	90.00 ± 4.08	82.35 ± 4.65	81.43 ± 2.95	92.12 ± 2.24	97.99 ± 2.39	96.45 ± 0.64	88.86 ± 2.46
SVM	-	92.43 ± 1.29	84.38 ± 0.42	89.00 ± 4.90	75.29 ± 3.98	75.33 ± 3.28	92.69 ± 1.30	99.10 ± 0.47	97.22 ± 0.21	93.43 ± 0.51

F.4 Average classification accuracy of GradEnFS with 20 random seeds

To further demonstrate the stability of our proposed method, we increase the number of random seeds from 5 to 20 to evaluate GradEnFS and present the results in Table 8. The SVM classification accuracy of GradEnFS averaged from 20 independent runs, remains similar to the results averaged from 5 independent runs, and still outperforms the classification accuracy of other methods, which were averaged from 5 independent runs on 6 out of 9 datasets (see Table 2).

Table 8: Classification Accuracy. The results are the SVM classification accuracy average over twenty independent runs (%).

Method\Dataset	BASEHOCK	PCMAC	Prostate_GE	TOX_171	Madelon	Isolet	COIL20	USPS	MNIST
GradEnFs_batch	89.03 ± 1.14	83.17 ± 1.41	88.25 ± 2.86	74.12 ± 7.12	76.08 ± 2.21	91.36 ± 2.05	98.80 ± 0.47	97.23 ± 0.26	83.46 ± 3.14
GradEnFS_epoch	89.85 ± 1.61	84.22 ± 2.40	87.75 ± 7.50	65.29 ± 6.14	77.19 ± 2.22	87.15 ± 2.46	98.59 ± 0.94	95.86 ± 0.42	93.25 ± 1.11

F.5 Comparison of GradEnFS and Lasso

Lasso is a widely used feature selection method that imposes a penalty on the absolute size of the coefficients in a linear regression model, promoting sparsity and leading to automatic feature selection. We conducted an experiment to compare our methods with Lasso on the quality of the selected feature subset and present the results in Table 9. The results demonstrate that GradEnFS outperforms Lasso in 6 out of 9 cases, particularly on Madelon and Isolet datasets. This superiority can be attributed to the ability of neural networks to learn nonlinear dependencies, while Lasso primarily operates with linear models. Additionally, sparsity reduces noise, and the network diversity achieved by dynamic sparse training strengthens the ensemble effect, further enhancing performance.

Table 9: Classification Accuracy. The results are the SVM classification accuracy average over five independent runs (%).

Method\Dataset	BASEHOCK	PCMAC	Prostate_GE	TOX_171	**Madelon**	Isolet	COIL20	USPS	MNIST
Lasso	90.65 ± 1.07	**89.59 ± 1.94**	**90.00 ± 2.00**	72.35 ± 2.35	59.07 ± 0.08	78.53 ± 1.13	96.04 ± 0.89	96.97 ± 0.22	**95.65 ± 0.26**
GradEnFS	**92.43 ± 1.29**	84.38 ± 0.42	89.00 ± 4.90	**75.29 ± 3.98**	**75.33 ± 3.28**	**92.69 ± 1.30**	**99.10 ± 0.47**	**97.22 ± 0.21**	93.43 ± 0.51

G Additional Details of the Algorithm's Convergence

To further investigate the convergence of the proposed algorithm, we present the SVM accuracies evaluated on selested feature subset per epoch for all datasets in Figure 10. The results show that most of the datasets achieve a stable, high-quality feature subset after a few epochs, further validating the proposed algorithm's fast convergence speed and stability of the selected subset. Moreover, we also present the validation losses per epoch of the sparse MLPs during training for all datasets in Figure 11. In this work, we utilize the SET algorithm to train sparse MLPs. Empirical evidence from the SET paper and subsequent studies has demonstrated that SET

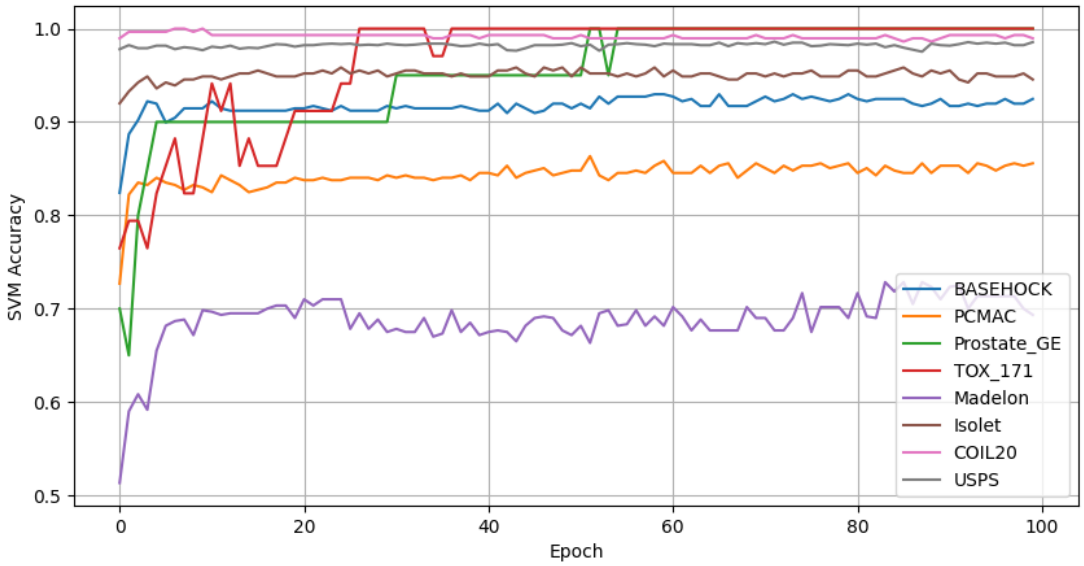

Figure 10: SVM accuracies on different epochs for all datasets.

does not hinder the convergence of models and can achieve convergence in MLPs and other network types (e.g., CNNs). SET updates weight matrices after standard feed-forward and backpropagation, including dropping weights with small magnitudes and adding zero connections, thus minimally impacting loss values. Figure 11 illustrates that using SET to train MLPs in our algorithm also leads to model convergence, indicating that the additional process of measuring neuron importance does not affect the convergence of the sparse model.

H MNIST Visualization

The visualization of the handwritten digits in the MNIST dataset is depicted in Figure 12.

I Additional Results of Visualization Heatmap

We compared the visualization heatmaps between GradEnFS and QS. The Quick Selection (QS) method, which leverages the SET algorithm for feature selection, can be hindered by the random regrow criterion within SET. This randomness often leads to slow convergence in the search for an optimal feature subset. However, because of SET's simplicity and energy-saving characteristics compared to other DST algorithms and the widespread success of applying SET to various ANNs, we still choose to combine the SET algorithm with the sensitivity-based dynamic neuron importance metric in our algorithm design. Therefore, we want to study whether the random regrow criterion will also affect the convergence speed of GradEnFS. To address this query, we run GradEnFS and QS on MNIST, and then we visualize the importance scores of input neurons as a heatmap at several epochs, which is shown in Figure 13. We also visualize the pixels (features) each algorithm selects after several epochs based on their importance metric and present the results in Figure 14.

Illustrated in Figure 13 and 14, following the initial epoch of neuron importance score calculation by GradEnFS, it already becomes evident that specific pixels positioned in the image's central region have higher neuron importance scores than the pixels at the edge of the image. This difference in importance scores between center and edge pixels continues growing until the fifth epoch. Interestingly, after the fifth epoch, the pixels recognized as informative by GradEnFS are not solely restricted to the image's center; they start to distribute across other areas, resulting in a distinct pattern of some important pixels gathered in the middle of the image and some

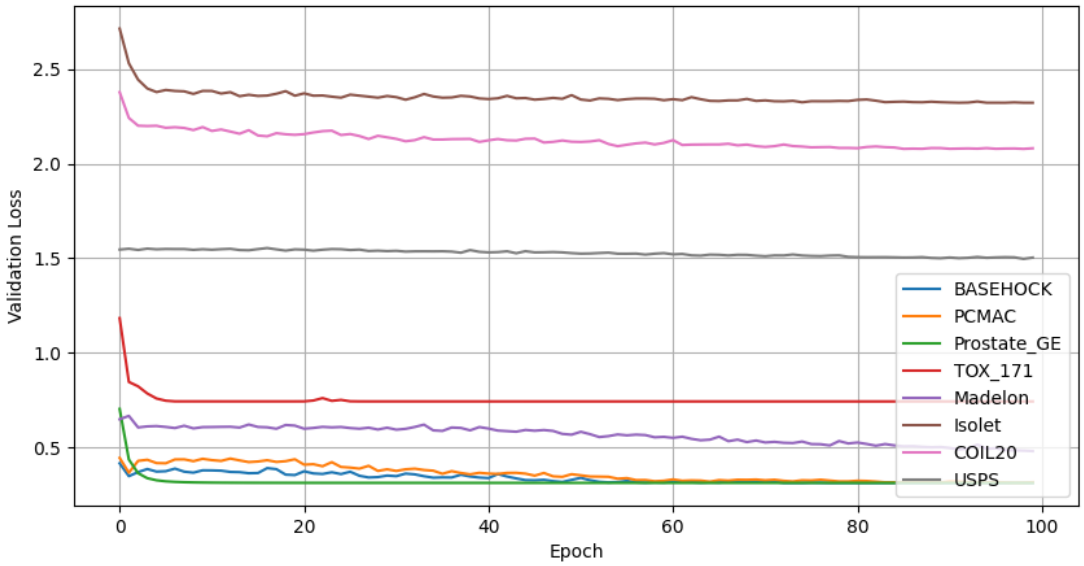

Figure 11: Validation losses per epoch of the sparse MLPs during training for all datasets.

important pixels surrounding this central cluster of significant pixels. This phenomenon is attributed to the algorithm's ability to recognize the pixels adjacent to the pixels forming the digit as significant contributors to the image's classification prediction, which means the model needs the edges of the digit to make a prediction. Notably, this distinct pattern had already emerged by epoch 20, and the visualization at epoch 50 closely resembles that of epoch 100, indicating that GradEnFS achieves rapid convergence from epoch 20. On the other hand, while QS ultimately converges to a pattern closely resembling GradEnFS after 100 epochs of training, it is noteworthy that QS only begins to exhibit higher neuron importance in the middle pixels compared to the neighboring pixels by the 20th epoch. This observation suggests that QS is more influenced by the random regrow criterion in SET, leading to a comparatively slower convergence rate when compared to GradEnFS.

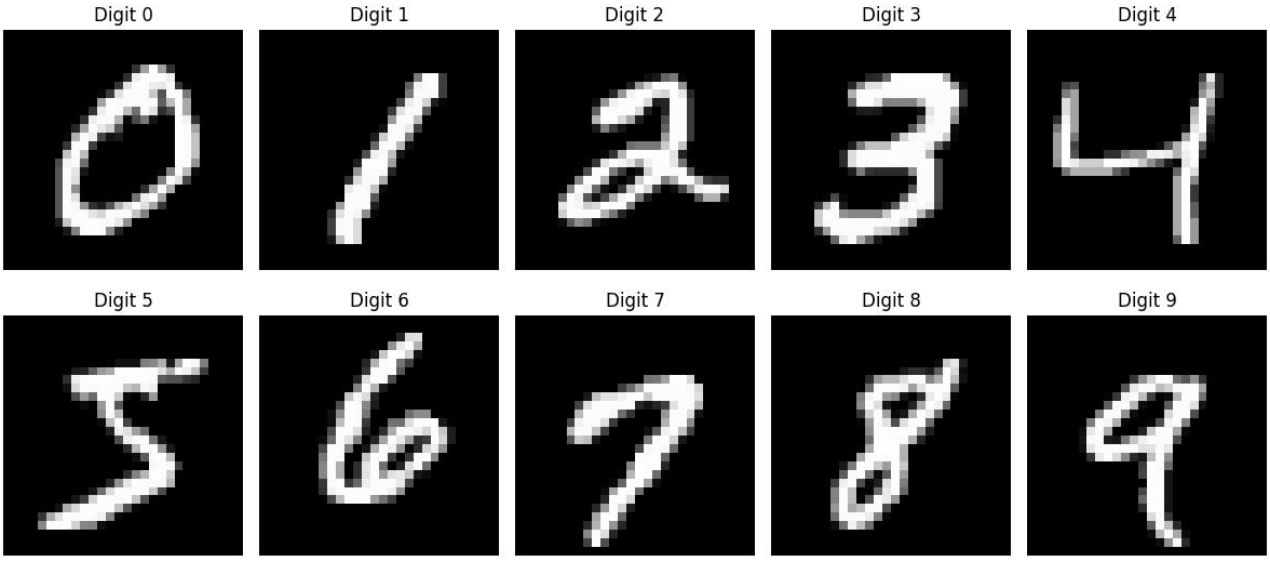

Figure 12: MNIST Dataset

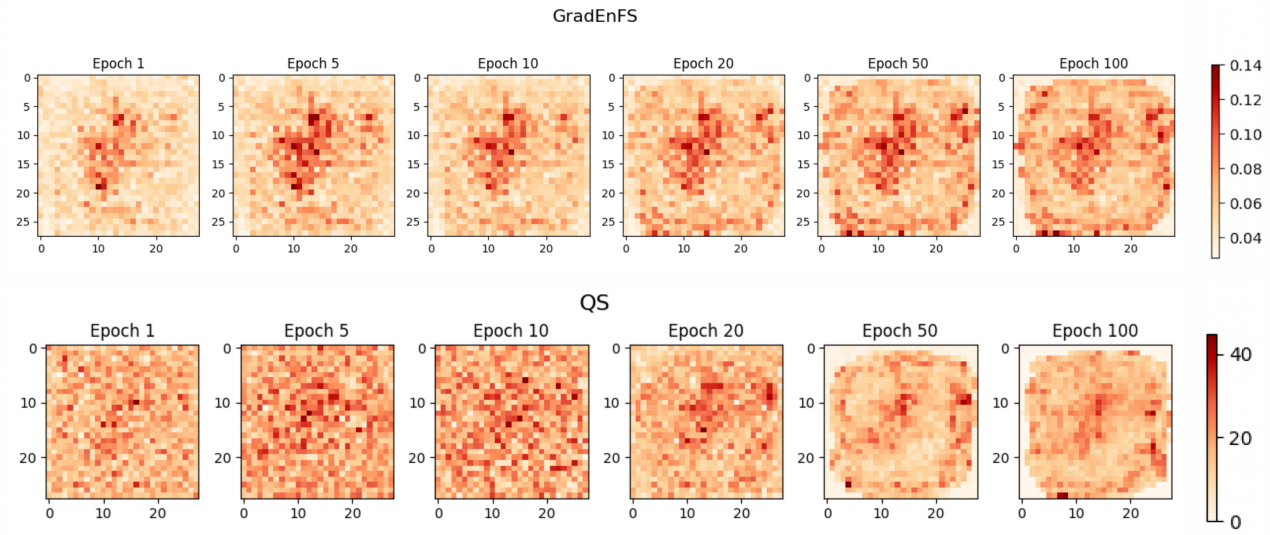

Figure 13: Neuron importance scores visualization of GradEnFS (above) and QS (below) after epoch 1, 5, 10, 20, 50 and 100.

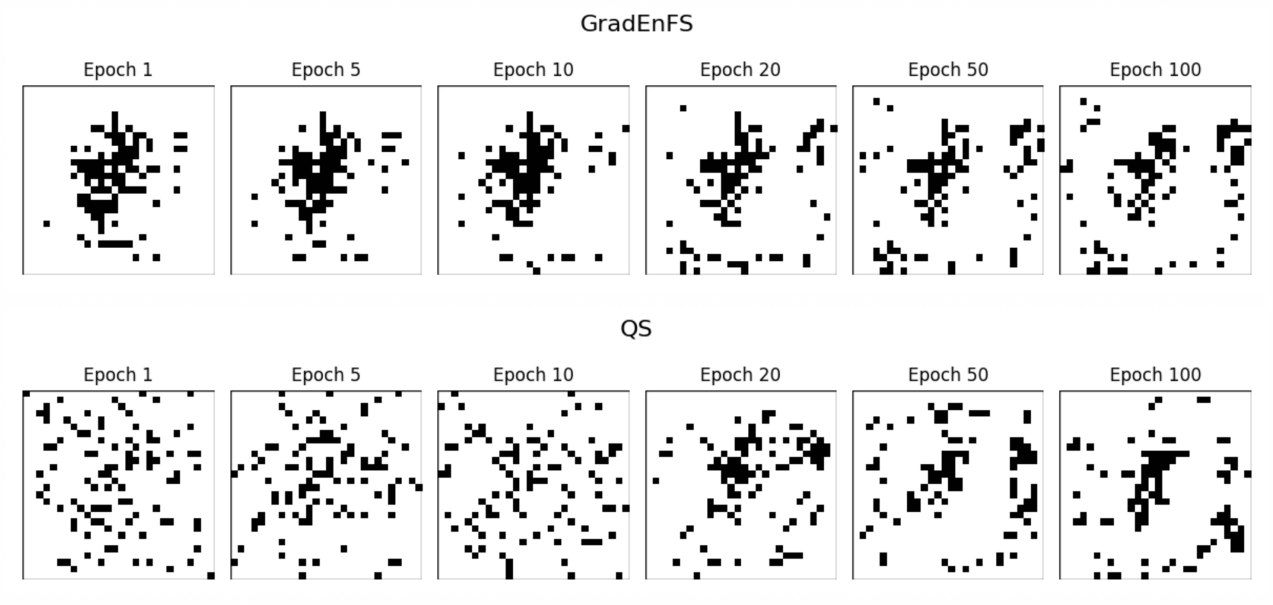

Figure 14: Selected pixels (features) visualization of GradEnFS (above) and QS (below) after epoch 1, 5, 10, 20, 50 and 100 (K=100).