Where to Pay Attention in Sparse Training for Feature Selection?

Ghada Sokar
Eindhoven University of Technology
g.a.z.n.sokar@tue.nl

Zahra Atashgahi
University of Twente
z.atashgahi@utwente.nl

Mykola Pechenizkiy
Eindhoven University of Technology
m.pechenizkiy@tue.nl

Decebal Constantin Mocanu
University of Twente
Eindhoven University of Technology
d.c.mocanu@utwente.nl

Abstract

A new line of research for feature selection based on neural networks has recently emerged. Despite its superiority to classical methods, it requires many training iterations to converge and detect informative features. The computational time becomes prohibitively long for datasets with a large number of samples or a very high dimensional feature space. In this paper, we present a new efficient unsupervised method for feature selection based on sparse autoencoders. In particular, we propose a new sparse training algorithm that optimizes a model's sparse topology during training to pay attention to informative features quickly. The attention-based adaptation of the sparse topology enables fast detection of informative features after a few training iterations. We performed extensive experiments on 10 datasets of different types, including image, speech, text, artificial, and biological. They cover a wide range of characteristics, such as low and high-dimensional feature spaces, and few and large training samples. Our proposed approach outperforms the state-of-the-art methods in terms of selecting informative features while reducing training iterations and computational costs substantially. Moreover, the experiments show the robustness of our method in extremely noisy environments[1].

1 Introduction

Feature selection plays a crucial role in data mining and machine learning tasks with the explosion in the size and dimensionality of real-world data [68; 10; 30; 44; 23]. It aims to efficiently select a subset of the features that are most informative and remove the irrelevant or redundant ones. It is useful for alleviating the curse of dimensionality, interpretability of model-driven decisions, generalization of downstream tasks, reducing the prohibitive memory and computational costs, and avoiding the expensive costs of collecting the full set of features as in biological studies [11; 38; 31; 52].

Many methods have been proposed for feature selection in supervised, semi-supervised, and unsupervised settings [75; 10; 60; 51; 19; 1]. Recently, using neural networks for feature selection has received increasing attention due to its power to learn non-linear dependencies among input features [39]. One limitation of current neural network-based methods is that they are computationally expensive. The computational costs result from training *dense* models for *many* training iterations until informative features are recognized. The costs increase for datasets with a very large number of training samples or high dimensional feature space. This limitation is recently addressed in [4] by utilizing sparse networks with dynamic sparsity for feature selection. An autoencoder is trained with

[1] Code is available at https://github.com/GhadaSokar/WAST.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

the sparse training algorithm SET [54], which explores different sparse topologies during training by dropping a portion of the connections and growing the same portion *randomly*. The importance of an input feature is estimated after convergence based on its outgoing sparse connections. Training sparse networks from scratch decreases memory and computational costs compared to previous dense-based methods. Yet, the *random* exploration of sparse topologies requires many training iterations to identify informative features.

In this paper, we propose an efficient unsupervised method for feature selection. We introduce a new sparse training algorithm for autoencoders that quickly detects the informative input features. Specifically, we optimize the sparse topology to learn **W**here to pay **A**ttention during **S**parse **T**raining. We named our method **WAST**. WAST exploits the information from the reconstruction loss and the weights of sparse connections to guide the topological search. It redistributes the sparse connections to spot informative features quickly. We evaluate our proposed approach on ten datasets from various domains, including image, speech, text, artificial, and biological. Through extensive experiments, we find that WAST identifies the informative features after a few training iterations and outperforms state-of-the-art unsupervised feature selection methods. Our main contributions are:

- We propose a new efficient unsupervised method for feature selection, named WAST. WAST optimizes the sparse topology of an autoencoder to detect the informative features quickly.
- We perform extensive experiments on 10 benchmarks that cover various types and characteristics. Experimental results show the effectiveness of our method over the state-of-the-art methods in terms of selecting the informative features.
- WAST reduces the computational time compared to neural network-based methods substantially by reducing training iterations and employing highly sparse neural networks.
- We illustrate the robustness of our method in extremely noisy environments and its effectiveness for datasets with very high dimensional feature space and a few training samples.

2 Related Work

2.1 Feature Selection

Many feature selection methods were introduced for the supervised, semi-supervised, and unsupervised settings depending on whether the data is labeled, partially labeled, or unlabeled, respectively [75; 10; 60; 1]. Typically, feature selection methods are divided into three categories: filter, wrapper, and embedded methods. Filter methods are independent of the learning algorithms. They use ranking techniques for providing scores to the features, such as Laplacian score [26]. Despite being fast, they do not consider the relationship between the features, which may result in the selection of redundant features [11]. In the supervised setting, filter methods consider the relation between the feature and the class label, such as fisher_score [22], CIEF [41], and ICAP [33]. Wrapper methods exploit the performance of a predictive model to evaluate the quality of a subset of the features [70; 12; 74]. They are more effective than filter methods, yet they are computationally expensive [39]. Embedded methods incorporate feature selection into the learning phase of another algorithm. Multi-Cluster Feature Selection (MCFS) [9] uses regularization to select the best features that keep the multi-cluster structure of the data. Unsupervised Discriminative Feature Selection (UDFS) [72] uses $l_{2,1}$ regularization and discriminative analysis to select the most discriminative features. Similarly, in the supervised setting, RFS [56] and L1_L21 [45] exploit $l_{2,1}$-norm to introduce feature sparsity.

Another direction under the embedded category has recently emerged. It uses neural networks to perform feature selection by learning the non-linear dependencies among input features [40]. The success of autoencoders as a tool for feature extraction encourages unsupervised methods to explore their power for feature selection [7; 6; 67; 25; 18]. AutoEncoder Feature Selector (AEFS) [25] combines autoencoder regression and group lasso tasks. Concrete Autoencoder (CAE) [5] learns a concrete selector layer (encoder) that selects stochastic linear combinations of input features during training. After training, it converges to the target number of features. Despite the high performance of these methods, the large number of iterations required to train dense models increases the computational costs significantly. The conceptually closest work to ours is the recent work, Quick Selection (QS) [4]. It trains a sparse autoencoder from scratch. During training, different sparse topologies are explored using the SET method [54]. After each *training epoch* (i.e., pass on the full data), a fraction of the connections with the least magnitude is dropped, and the same fraction

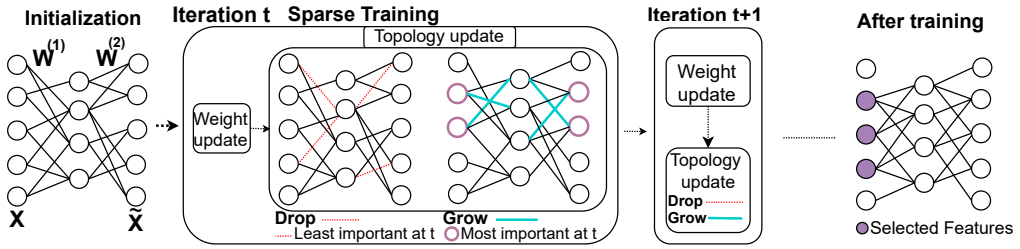


Figure 1: An overview of our proposed method WAST for unsupervised feature selection. A sparse autoencoder is initialized with uniformly distributed sparse connections. During sparse training, the connections are redistributed in the most important neurons at iteration t during the "drop-and-grow" cycle. After convergence, neurons with the highest importance are selected as the most informative features.

is randomly regrown. Few works based on multi-layer perceptron networks are proposed for the supervised setting [37; 61; 71]. In section 4, we denote non Neural Network-based (NN-based) methods as classical methods.

2.2 Sparse Training

Sparse training with dynamic sparsity, also known as Dynamic Sparse Training (DST), is a recent research direction that aims at accelerating the training of neural networks without sacrificing performance [27; 54; 47]. A neural network is initialized with a random sparse topology from *scratch*. The sparse topology (connectivity) and the weights are jointly optimized during training. During training, the sparse topology is changed periodically through a *drop-and-grow* cycle, where a fraction of the parameters are dropped, and the same portion is regrown among different neurons. An update schedule determines the frequency of topology updates. Many works have been proposed, focusing on improving the performance of sparse training for supervised image classification tasks by introducing different criteria for connection regrowth [20; 34; 59; 49; 73; 50]. DST demonstrated its success in other fields as well, such as continual learning [62], feature selection [4], ensembling [46], federated learning [77; 8], adversarial training [57], and deep reinforcement learning [63].

3 Where to pay Attention in Sparse Training (WAST)?

Problem Formulation. Let $\mathcal{D}$ be a dataset with samples $\{\mathbf{x}^{(j)}\}_1^n$, where $\mathbf{x}^{(j)} \in \mathbb{R}^m$, and m and n are the number of features and samples, respectively. The goal is to select a subset of the features of a size K in an *unsupervised* manner, which is most representative of the whole feature space m.

WAST is a new sparse training method for neural network-based unsupervised feature selection. The basic idea of WAST is to pay attention to input features based on their estimated importance during training to detect the most informative features quickly. An overview of WAST is shown in Figure 1. We use a *sparse* autoencoder with a *single* hidden layer of h neurons. Let $f_{\mathbf{W}}$ be an autoencoder model parameterized by sparse weights $\mathbf{W} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$. The network is initialized with a certain sparsity level $s = 1 - \frac{\|\mathbf{W}^{(1)}\|_0}{m \times h}$, where $\|.\|_0$ is the L_0 norm. The sparsity level is kept fixed during training. Initially, the sparse weights are uniformly distributed on the neurons. During training, besides weights optimization, the sparse topology is optimized such that the sparse connections are gradually redistributed on the most informative features. To achieve this goal, we propose new criteria to update the sparse topology and a new update schedule, as we will explain next.

Training. An autoencoder model is trained to minimize the reconstruction loss. We use mean squared error to measure the loss between an input sample $\mathbf{x}$ and the reconstructed one $\tilde{\mathbf{x}}$ as follows:

$$L = \|\tilde{\mathbf{x}} - \mathbf{x}\|_2^2 \,, \tag{1}$$

where $\tilde{\mathbf{x}} = f_{\mathbf{W}}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$, and σ is a non-linear activation function. After each weight update using a batch of the data, we adapt the sparse topology. The effect of the update schedule on performance is shown in Appendix E. The sparse topology is updated through a **drop-and-grow** cycle.

Algorithm 1 WAST

Input: Autoencoder network $f_{\mathbf{W}}$ with one hidden layer of size h, Dataset $\mathcal{D}$ with m features
Input: Learning rate η, target number of features K, sparsity level s, λ, α
Output: Indices of selected features from the m features ($\mathcal{R}$)
Initialize $\mathbf{W} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$ with sparsity level s
$\mathcal{I}_{I_i} \leftarrow 0 \quad \mathcal{I}_{O_i} \leftarrow 0 \quad \forall i \in \{1, ..., m\}$
for each training step t **do**
  Sample a Batch $B_t \sim \mathcal{D}$ with size b
  $\tilde{\mathbf{x}}^{(j)} \leftarrow f_{\mathbf{W}}(\mathbf{x}^{(j)}), \quad \forall \mathbf{x}^{(j)} \in B_t$
  $L \leftarrow \frac{1}{b} \sum_{j=0}^{b} \left\| \tilde{\mathbf{x}}^{(j)} - \mathbf{x}^{(j)} \right\|_2^2$
  $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L$
  $\mathcal{I}_{O_i} \leftarrow \mathcal{I}_{O_i} + \lambda |\frac{\partial L}{\partial \tilde{\mathbf{x}}_i}| + (1-\lambda) \sum_{k=1}^{h} |\mathbf{W}_{ki}^{(2)}| \quad \forall i \in \{1, ..., m\}$
  $\mathcal{I}_{I_i} \leftarrow \mathcal{I}_{I_i} + \lambda |\frac{\partial L}{\partial \tilde{\mathbf{x}}_i}| + (1-\lambda) \sum_{k=1}^{h} |\mathbf{W}_{ik}^{(1)}| \quad \forall i \in \{1, ..., m\}$
  $\mathbf{W}^{(l)} \leftarrow$ Drop $(|\mathbf{W}^{(l)}|, \mathcal{I}_O, \mathcal{I}_I, \alpha) \quad \forall l \in \{1, 2\}$
  $\mathbf{W}^{(l)} \leftarrow$ Grow $(\mathcal{I}_O, \mathcal{I}_I, \alpha) \quad \forall l \in \{1, 2\}$
end for
$\mathcal{R} \leftarrow top(\mathcal{I}_I, K)$

A fraction α of the connections with the least importance is dropped from each layer $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$, and the same portion of new connections are regrown. New connections are initialized to zero.

The main contribution of this work lies in where the new connections are regrown. Different from the previous method (QS) [4], where the connections are randomly regrown, we optimize the topology to pay fast attention to informative features during training. To this end, we measure the importance of each neuron in the input and output layers by leveraging the information available during training. The importance of a neuron is estimated by its impact on the reconstructed loss and the magnitude of its connected weights.

Specifically, we measure how sensitive the reconstruction loss L is to changes in the reconstructed output $\tilde{\mathbf{x}}$. The first-order approximation for the change in loss with respect to a small perturbation $\delta = \{\delta_i\}$ in the output $\tilde{\mathbf{x}} = \{\tilde{\mathbf{x}}_i\}$ can be written as a sum of its individual components as follows:

$$L(\tilde{\mathbf{x}} + \delta) - L(\tilde{\mathbf{x}}) \approx \sum_{i=1}^{m} \frac{\partial L}{\partial \tilde{\mathbf{x}}_i} \delta_i, \tag{2}$$

where $\frac{\partial L}{\partial \tilde{\mathbf{x}}_i}$ is the gradient of the loss with respect to the output neuron $\tilde{\mathbf{x}}_i$. We define the importance of a neuron i in the output layer at training iteration t as:

$$\mathcal{I}_{O_i}^{(t)} = \mathcal{I}_{O_i}^{(t-1)} + \lambda |\frac{\partial L}{\partial \tilde{\mathbf{x}}_i}| + (1-\lambda) \sum_{j=1}^{h} |\mathbf{W}_{ji}^{(2)}|, \tag{3}$$

where $|\mathbf{W}_{ji}^{(2)}|$ is the magnitude of the incoming connection from neuron j to neuron i and λ is a hyperparameter coefficient to balance the two components (See Appendix A). The same criterion is applied for the input neurons ($\mathcal{I}_{I_i}^{(t)}$), except that the magnitude of input weights ($|\mathbf{W}^{(1)}|$) is used instead of the output weights. This criterion considers two cases: the information from large gradients, especially at the beginning of the training, and the large weights resulting from the optimization at later stages. The new connections in the input and output layers are regrown on the neurons with the highest importance. For instance, the new connections in the input layer are given by $top_{i \notin \tilde{\mathbf{W}}^{(1)}}(\mathcal{I}_I \times \mathcal{I}_H^\top, r)$, where $top(Q, k)$ gives the indices of the top-k elements in a matrix Q, $\mathcal{I}_H$ is the importance of the hidden neurons, $\tilde{\mathbf{W}}^{(1)}$ is the set of remaining weights after the drop phase, and r is the number of regrown connections based on the fraction α. We consider the neurons in the hidden layer to be equally important.

Table 1: The characteristics of datasets.

Type	Dataset	#Features	#Classes	#Samples	#Train	#Test
Artificial	Madelon [24]	500	2	2600	2000	600
Image	UPSP [32]	256	10	9298	7438	1860
	COIL-20 [55]	1024	20	1440	1152	288
	MNIST [36]	784	10	70000	60000	10000
	Fashion MNIST [69]	784	10	70000	60000	10000
Speech	Isolet [21]	617	26	7737	6237	1560
Time Series	HAR [2]	516	6	10299	7352	2947
Text	PCMAC [35]	3289	2	1943	1544	389
Biological	SMK-CAN-187 [64]	19993	2	187	149	38
	GLA-BRA-180 [65]	49151	4	180	144	36

During the drop phase, the importance of a connection w in the input (output) layer is estimated by its magnitude and the importance of its connected input (output) neuron ($\mathcal{I}_i^{(t)}$) as follows:

$$\mathcal{I}_w^{(t)} = |w|\mathcal{I}_i^{(t)}, \tag{4}$$

where $\mathcal{I}_i^{(t)}$ takes the value of $\mathcal{I}_{I_i}^{(t)}$ and $\mathcal{I}_{O_i}^{(t)}$ for the input and output layers, respectively. The effect of each component in the neuron and connection importance criteria is studied in Section 5.3. The periodic update of the sparse topology redistributes the connections on the effective features.

Feature Selection. After training, we select the top-K neurons with the highest importance from the input layer as the most informative K features. The details of WAST are provided in Algorithm 1.

4 Experiments

4.1 Baselines

We compared our method with several autoencoder-based (QS [4], CAE [5], AEFS [25]) and classical methods (lap_score [26], MCFS [9], DUFS [42]) for unsupervised feature selection. On top of that, we include comparisons with seven state-of-the-art supervised feature selection methods. Although the latter is not the focus of this paper and not a typical comparison in the literature, we prefer to show that WAST performs competitively also with the supervised methods while being computationally efficient and applicable for cases where labels are not available or expensive to collect.

4.2 Datasets

We evaluate our method on 10 publicly available datasets, including image, speech, text, time series, biological, and artificial data. They have a variety of characteristics, such as low and high-dimensional features and a small and large number of training samples. Details are in Table 1.

4.3 Experimental Settings

Evaluation Metrics. An efficient feature selection method should have high learning accuracy with less memory and computational costs [10]. Hence, we evaluate multiple metrics. **(1) Classification Accuracy**: It is typically evaluated by a machine learning model [10]. We trained a classifier using the K selected features by the studied methods. Here, we use one of the most popular classifiers, support vector machines (SVM) [14]. Other classifiers could be used. Yet, we choose to use a non-NN-based classifier for evaluation to avoid any biased advantages towards the NN-based baselines over the others. We studied the performance of 6 different values of the number of selected features (K). We report the average accuracy of the test data over 5 seeds. **(2) Memory cost:** We calculate the number of network parameters (#params) used by each NN-based method. **(3) Computational cost:** We calculate the number of Floating-point operations (FLOPs) consumed to train a neural network. See Appendix A.2 for details.

Implementation. We implemented WAST and QS [4] with PyTorch [58] (see Appendix H). For CAE [5] and AEFS [25], we used the code provided by the authors of CAE with MIT license[2]. For DUFS

[2]https://github.com/mfbalin/Concrete-Autoencoders

Table 2: Classification accuracy (%) (↑) using unsupervised and supervised feature selection methods. 50 features are selected for all datasets except Madelon, where 20 features are used. The best performer from the unsupervised methods is indicated in bold font, while the best performer from the supervised methods is in blue.

		Method	Madelon	USPS	COIL-20	MNIST	FashionMNIST
Unsupervised	Classical	lap_score [26]	49.50±0.00	70.54±0.00	78.12±0.00	23.94±0.00	27.07±0.00
		MCFS [9]	51.83±0.00	93.33±0.00	97.22±0.00	-	-
		DUFS [42]	52.57±1.35	95.62±0.54	97.43±1.22	62.09± 0.00	74.69±1.86
	NN-based	AEFS [25]	60.16±4.61	95.86±0.48	99.48±0.41	93.22±1.38	80.88±0.71
		CAE [5]	80.90±2.86	95.04±0.59	94.54±2.92	**96.20±0.14**	**84.66±0.16**
		QS [4]	82.07±1.10	95.88±0.31	99.17±0.42	94.07±0.04	82.65±0.38
		WAST (ours)	**83.27±0.63**	**96.69±0.27**	**99.58±0.14**	95.27±0.26	82.16±0.57
Supervised	Classical	Fisher_score [22]	75.67±0.00	91.02±0.00	88.89±0.00	86.11±0.00	67.85±0.00
		L1_L21 [45]	49.33±0.00	86.99±0.00	92.01±0.00	62.26±0.00	69.57±0.00
		CIFE [41]	54.50±0.00	61.29±0.00	59.38±0.00	89.30±0.00	66.86±0.00
		ICAP [33]	78.00±0.00	95.22±0.00	99.31±0.00	89.03±0.00	59.52±0.00
		RFS [56]	83.00±0.00	95.32±0.00	66.32±0.00	-	-
	NN-based	LassoNet [37]	79.50±1.22	95.80±0.12	95.83±1.18	94.38±0.12	82.63±0.23
		STG [71]	59.53±1.90	95.78±0.60	97.57±1.70	92.53±0.86	83.32±0.45

		Method	Isolet	HAR	PCMAC	SMK	GLA
Unsupervised	Classical	lap_score [26]	75.71±0.00	82.80±0.00	49.87±0.00	81.58±0.00	66.67±0.00
		MCFS [9]	81.41±0.00	80.29±0.00	53.47±0.00	78.95±0.00	75.00±0.00
		DUFS [42]	**85.62±2.53**	86.90±1.06	57.79±3.18	81.05±3.07	70.83±1.39
	NN-based	AEFS [25]	80.94±2.02	89.54±0.44	60.40±2.37	79.48±3.07	67.76±6.21
		CAE [5]	78.90±1.24	86.26±2.41	55.08±0.00	78.94±2.37	70.56±4.50
		QS [4]	74.62±2.12	89.68±0.38	55.78±3.25	81.58±3.72	68.89±4.78
		WAST (ours)	85.33±1.39	**91.20±0.16**	**60.51±2.53**	**84.74±1.05**	**75.56±4.08**
Supervised	Classical	Fisher_score [22]	75.64±0.00	83.68±0.00	86.38±0.00	73.68±0.00	63.89±0.00
		L1_L21 [45]	55.90±0.00	81.30±0.00	53.98±0.00	84.21±0.00	69.44±0.00
		CIFE [41]	59.81±0.00	84.15±0.00	75.84±0.00	81.58±0.00	58.33±0.00
		ICAP [33]	75.06±0.00	88.70±0.00	87.66±0.00	73.68±0.00	72.22±0.00
		RFS [56]	77.31±0.00	88.23±0.00	67.61±0.00	76.32±0.00	-
	NN-based	LassoNet [37]	85.70±0.38	93.93±0.15	86.53±1.25	77.37± 3.57	76.67±2.22
		STG [71]	89.38±1.19	91.75±0.59	56.04±1.90	81.05±1.29	71.11±2.83

[42], LassoNet [37], and STG [71], we used the official codes with MIT license[3][4][5]. For classical baselines, we used the Scikit-Feature library with GNU General Public license [39][6]. NN-based and classical methods are trained on Nvidia GPUs and CPUs, respectively. We consider a 12 hours limit on the running time of each experiment. Experiments that exceed this limit are not considered.

Implementation Details. For all NN-based methods except CAE [5], we use a single hidden layer of 200 neurons. The architecture of CAE consists of two layers. The size of the hidden layers is dependent on the chosen K; $[K, \frac{3}{2}K]$. For WAST and QS, we use a sparsity level of 0.8. Following [4], we report the accuracy of NN-based baselines after 100 epochs unless stated otherwise. Note that some baselines reported a higher number of epochs (e.g., CAE [5] uses 200 epochs for some datasets). Yet, we keep 100 epochs for all cases for a fair comparison. For WAST, we train the model for 10 epochs. Following [4], we add a Gaussian noise with a factor of 0.2 to the input in WAST and QS [4]. Details of the hyperparameters are in Appendix A.1.

4.4 Results

Accuracy. Table 2 shows the classification accuracy of the studied datasets. Here, we report the challenging case where a few best informative features have to be selected. We use a K of 50 for all datasets except for Madelon, where a K of 20 is used as the remaining 480 features are pure noise. Experiments with various values for $K \in \{25, 50, 75, 100, 150, 200\}$ can be found in Appendix B, with a summary provided next.

Unsupervised NN-based methods outperform the classical ones on all datasets except one, where the performance difference is marginal. WAST outperforms unsupervised NN-based methods on

[3]https://github.com/Ofirlin/DUFS

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

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

[6]https://jundongl.github.io/scikit-feature/

Table 3: Memory and computational costs (↓) estimated by the #params and FLOPs (10^{12}), respectively, for NN-based methods. Unlike other methods, the architecture of CAE [5] is dependent on the value of K. Here, we report the costs for $K = 50$.

		(Fashion)MNIST		USPS		COIL-20		Isolet	
Method	Type	#params	FLOPs	#params	FLOPs	#params	FLOPs	#params	FLOPs
AEFS[25]	Dense	313600	11.28	102400	0.45	409600	0.28	246800	0.92
CAE[5]	Dense	101750	3.66	35750	0.15	131750	0.09	80875	0.30
QS[4]	Sparse	**62720**	2.25	**20480**	0.09	**81920**	0.05	**49360**	0.18
WAST	Sparse	**62720**	**0.22**	**20480**	**0.009**	**81920**	**0.005**	**49360**	**0.01**
		HAR		PCMAC		SMK		GLA	
AEFS[25]	Dense	206400	0.91	1315600	1.22	7997200	0.71	19660400	1.69
CAE[5]	Dense	68250	0.30	414875	0.38	2502875	0.22	6147625	0.53
QS[4]	Sparse	**41280**	0.18	**263120**	0.24	**1599440**	0.14	**3932080**	0.33
WAST	Sparse	**41280**	**0.01**	**263120**	**0.02**	**1599440**	**0.01**	**3932080**	**0.03**

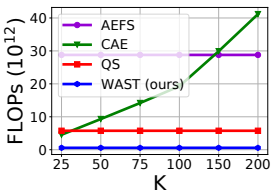


Figure 2: FLOPs (↓) required for all datasets except Madelon at different values of K.

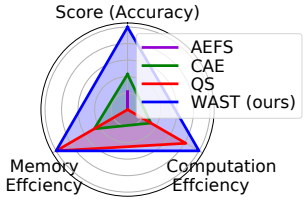


Figure 3: Comparison using various evaluation metrics computed on all datasets and 6 different values for K.

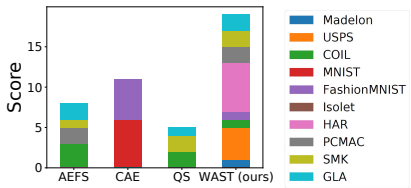


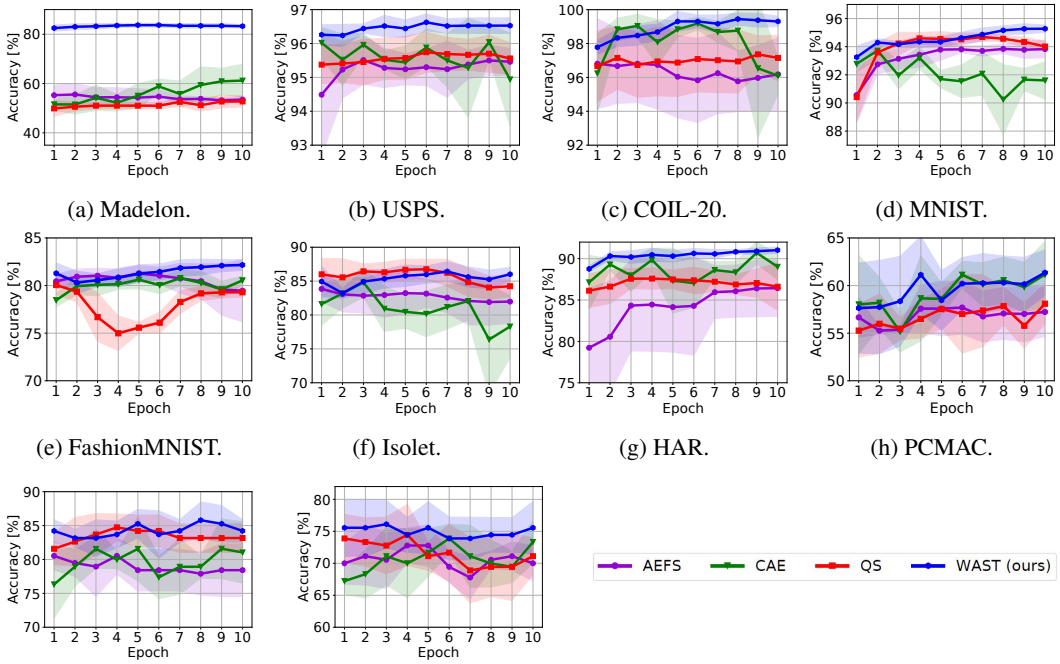
Figure 4: Score (↑) (how many times a method is the best performer) of different methods on each dataset for 6 different values of K.

all datasets except MNIST and Fashion MNIST, where CAE is the best performer. It is worth emphasizing that the superior performance of WAST is achieved with a 90% reduction in the number of training iterations. Supervised baselines outperform the classical supervised ones in 6 cases. Interestingly, WAST achieves a competitive performance to the supervised baselines. It outperforms the best-supervised performer in 5 cases.

Memory and Computational Costs. Table 3 shows the memory and computational costs consumed by each unsupervised NN-based method. The high memory and computational costs are caused by the high number of training samples, as in MNIST, or the high-dimensional feature space, as in GLA. Training a sparse model from scratch, as in WAST and QS, reduces these costs substantially. QS reduces the computational cost by 80%. Interestingly, besides the reduction resulting from using sparse neural networks, WAST reduces the computational cost further to 98% due to the fast identification of the informative features. Both WAST and QS reduce the network size by 80%.

The network size of CAE is dependent on the target value of K. This has two limitations: training the model for every target K and the increase of the memory and computational costs with higher values for K. On the other hand, WAST derives the importance of each feature with a single training run independently of the target K. This is illustrated in Figure 2. The figure shows the accumulated FLOPs for all datasets except Madelon at each value of the studied K.

Overall Performance. To give a holistic view of the performance of each method across all dimensions (accuracy, memory cost, and computational cost) on all datasets and all studied values of K, we calculate the performance in each case. Figure 3 illustrates the normalized performance using min-max scaling for all cases. The "score" metric represents the number of times a method is the best performer in terms of accuracy. Details are provided in Appendix A.2. CAE has a higher score than AEFS, while the former is the best only for image datasets with a large number of samples (i.e., MNIST and Fashion MNIST), as shown in Figure 4. AEFS has the highest memory and computational costs. QS improves the memory and computational costs but has a lower score. WAST has the highest score and spans more dataset *types* with different *characteristics* (e.g., high dimensional data) than the baselines (Figure 4). The performance gain is accompanied by a significant improvement in memory and computational efficiency. Detailed results are provided in Appendix B.



(a) Madelon. (b) USPS. (c) COIL-20. (d) MNIST.

(e) FashionMNIST. (f) Isolet. (g) HAR. (h) PCMAC.

(i) SMK. (j) GLA.

Figure 5: The performance of different methods at the early stages of the training. The test accuracy is reported for at first 10 training epochs using K of 50 for all datasets except Madelon, where K of 20 is used.

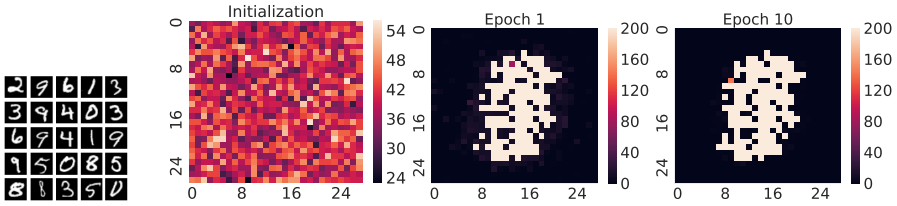
5 Analysis

5.1 Effect of Fast Attention During Training

To analyze the impact of attention to the informative features in WAST during training, we study the learning behavior of all unsupervised NN-based methods on the test data for the first 10 epochs. Since CAE has a parameter that is dependent on the max number of epochs, we performed ten separate runs and varied the max number of epochs each time. Figure 5 shows the accuracy using K of 50 for all datasets except Madelon, where K of 20 is used. The analysis reveals the following findings: **(1) Robustness to noisy environments:** The results on Madelon illustrate the robustness of WAST against the noisy features (480 out of 500). The figure shows the performance gap between WAST and the baseline methods starting from epoch 1. After 10 epochs, WAST outperforms the second-best performer by 22%. More experiments on noisy enviroments can be found in Appendix G. **(2) Performance on datasets with a large number of samples:** WAST outperforms CAE by 3.6% and 1.6% on MNIST and FashionMNIST, respectively, although CAE was the best performer on these datasets after training it for a larger number of epochs (Table 2). **(3) Consistency across different dataset types and characteristics:** We observe that some baselines achieve high performance in some domains while being worse in others. WAST has the highest consistency on different dataset types and characteristics, outperforming all baselines after 10 epochs (Appendix C). **(4) Stability:** WAST is more stable at the early stage of the training. This is represented by the standard deviation across different training runs (shaded region).

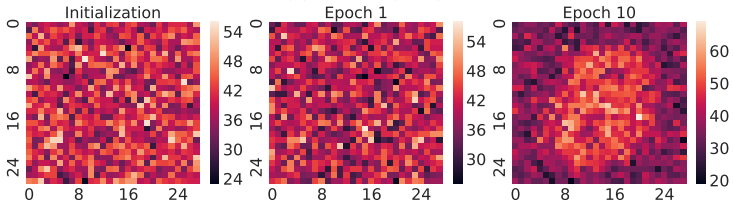
5.2 Visualization

Figure 6 illustrates how the sparse topology changes during training in WAST and QS. We performed this analysis on MNIST, where handwritten digits are centered in 28×28 grayscale images (Figure 6a). Initially, the sparse connections are uniformly distributed on the input neurons. With the guided adaptation of a sparse topology via attention, WAST detects the informative features quickly after one



(a) MNIST

(b) WAST (ours).



(c) QS [4].

Figure 6: Visualization. (a) Samples of the MNIST dataset. Each sample is 28×28, and the informative pixels are centered in the middle. The distribution of the sparse connections during training (i.e., the number of outgoing connections from each input neuron) on MNIST using WAST (b) and the QS method [4] (c). WAST has faster attention to the informative features after one epoch.

Table 4: Ablation study. Effect of each component in the criteria of neuron importance and connection importance. The accuracy (%) is reported using K of 20 and 50 on Madelon and others, respectively.

Criteria	#	Method	Madelon	USPS	HAR	PCMAC	SMK
Neuron	1	"w/o $\lvert\frac{\partial L}{\partial \tilde{\mathbf{x}}}\rvert$"	55.50±2.45	95.91±0.64	87.88±1.79	55.58±2.78	84.21±3.16
Neuron	2	"w/o $\lvert\mathbf{W}\rvert$"	82.47±0.75	95.56±0.18	**92.20±0.50**	58.66±4.18	75.26±2.11
Neuron	3	"w/o momentum"	81.60±1.06	96.60±0.40	90.74±0.94	58.30±4.80	78.95±3.33
Connection	4	"w/o $\mathcal{I}_i$"	83.27±0.63	96.54±0.17	90.11±0.52	55.53±3.07	84.74±1.05
		WAST (ours)	**83.27±0.63**	**96.69±0.27**	91.20±0.20	**60.51±2.53**	**84.74±1.05**

epoch. On the other hand, the random growth of the connections in QS requires more training epochs. The effect of increasing the frequency of topological exploration in QS is studied in Appendix F.

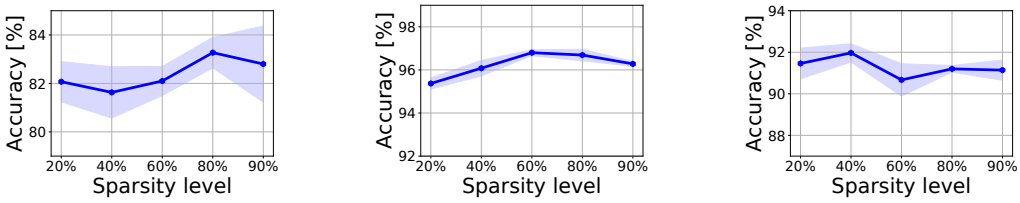
5.3 Ablation Study

We performed an ablation study to assess the contribution of each component in the proposed criteria of neuron and connection importance in the performance of WAST.

Neuron Importance. (#1) "w/o $\lvert\frac{\partial L}{\partial \tilde{\mathbf{x}}}\rvert$": Using only the magnitude of connected weights to determine the importance of a neuron (i.e., $\lambda = 0$), (#2) "w/o $\lvert\mathbf{W}\rvert$": Using only the sensitivity of the neuron to the change in the loss (i.e., $\lambda = 1$), and (#3) "w/o momentum": The importance of a neuron is based only on the *current* estimate of its sensitivity to the loss and currently connected weights.

Connection Importance. (#4) "w/o $\mathcal{I}_i$": The importance of a connection is estimated by its magnitude without considering the importance of its connected neuron $\mathcal{I}_i$.

We performed this analysis on datasets of different types. The results are summarized in Table 4. We observe that removing the sensitivity of a neuron to the loss (#1) has the biggest effect on performance. It has the highest impact on noisy environments, where it reduces the performance by 27.77% on Madelon. On the other datasets, it results in a $0.53 - 4.93\%$ reduction in the performance. Excluding the magnitude of the weights (#2) affects mainly the datasets with few training samples (e.g., SMK). Neglecting the neuron importance in the previous training iterations (#3) reduces the performance by $0.09 - 5.79\%$. Aligned with many prior works [20; 54; 27], we find that the weight magnitude is an effective metric for estimating the connection importance (#4). Moreover, our results show that enhancing it with the importance of its connected neuron improves the performance even further on some datasets without diminishing it on the others.



(a) Madelon. (b) USPS. (c) HAR.

Figure 7: The performance of WAST using different sparsity levels for the autoencoder model. The test accuracy is reported using K of 50 except on Madelon, where $K = 20$.

5.4 Effect of the Sparsity Level

We further study the effect of the sparsity level of the auto-encoder model on the performance of WAST. We evaluated 5 different sparsity levels $\{20\%, 40\%, 60\%, 80\%, 90\%\}$. All other settings are the same as the ones stated in Appendix A.

Figure 7 illustrates the classification accuracy using $K = 20$ for Madelon and K=50 for other datasets. We observe the performance of WAST is robust to the sparsity level. Yet, we mostly care about the performance at high sparsity levels, as the goal is to achieve high performance at a low computational cost.

6 Discussion

Conclusion. In this paper, we propose WAST, a new efficient neural network-based method for unsupervised feature selection. We train a sparse autoencoder from scratch and optimize the sparse topology during training to detect the informative features quickly. We performed extensive experiments in which we evaluated 55 cases on various datasets and different values of the number of selected features. WAST achieves the best performance on 19 cases, while the second-best unsupervised performer has a score of 11 cases. More interestingly, the superior performance is achieved with a few training iterations. WAST reduces the memory and computational costs by 80% and 98%, respectively. Moreover, we show the robustness of WAST towards very noisy environments, outperforming the state-of-the-art methods by 22% with limited training iterations. Finally, we show that WAST performs competitively with supervised-based methods, outperforming them on image datasets. This demonstrates the promise of adapting WAST in the supervised setting and motivates new sparse training algorithms for supervised and unsupervised feature selection.

Limitations. This work is a step toward exploiting the power of neural networks for feature selection in a computationally efficient manner. Besides the improvement in the accuracy of selecting the informative feature, fast attention to the important features during training reduces the number of training iterations substantially. Yet, like most sparse training methods in the literature, this proof-of-concept does not fully utilize the memory and computational advantages of sparse neural networks. This is due to the lack of hardware support for sparsity and the higher focus of the community on the algorithmic side [27]. Nevertheless, there is recent growing attention to hardware and software support for sparsity [28; 76; 66; 48; 15; 3; 13] (See Appendix H for discussion). This enables the pure sparse implementation of the method in the future.

Societal Impacts. With the emergence of big data, systems that can quickly select informative features and remove redundant ones become crucial. Besides the performance gain that could be achieved in the downstream tasks by removing irrelevant features, reducing the number of features improves memory and computational efficiency significantly. This enables providing energy-efficient systems. Moreover, it is useful for improving the interpretability of model-driven decisions. The *unsupervised* selection of the informative features is effective in cases where labeled data is limited or very expensive to collect. We do not expect that there is a negative societal impact.

References

[1] Ang, J.C., Mirzal, A., Haron, H., Hamed, H.N.A.: Supervised, unsupervised, and semi-supervised feature selection: a review on gene selection. IEEE/ACM transactions on computational biology and bioinformatics **13**(5), 971–989 (2015)

[2] Anguita, D., Ghio, A., Oneto, L., Parra Perez, X., Reyes Ortiz, J.L.: A public domain dataset for human activity recognition using smartphones. In: Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning. pp. 437–442 (2013)

[3] Ashby, M., Baaij, C., Baldwin, P., Bastiaan, M., Bunting, O., Cairncross, A., Chalmers, C., Corrigan, L., Davis, S., van Doorn, N., et al.: Exploiting unstructured sparsity on next-generation datacenter hardware (2019)

[4] Atashgahi, Z., Sokar, G., van der Lee, T., Mocanu, E., Mocanu, D.C., Veldhuis, R., Pechenizkiy, M.: Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. Machine Learning pp. 1–38 (2021)

[5] Balın, M.F., Abid, A., Zou, J.: Concrete autoencoders: Differentiable feature selection and reconstruction. In: International conference on machine learning. pp. 444–453. PMLR (2019)

[6] Bengio, Y.: Learning deep architectures for AI. Now Publishers Inc (2009)

[7] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)

[8] Bibikar, S., Vikalo, H., Wang, Z., Chen, X.: Federated dynamic sparse training: Computing less, communicating less, yet learning better. Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI) (2022)

[9] Cai, D., Zhang, C., He, X.: Unsupervised feature selection for multi-cluster data. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 333–342 (2010)

[10] Cai, J., Luo, J., Wang, S., Yang, S.: Feature selection in machine learning: A new perspective. Neurocomputing **300**, 70–79 (2018)

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

[12] Chen, J., Stern, M., Wainwright, M.J., Jordan, M.I.: Kernel feature selection via conditional covariance minimization. Advances in Neural Information Processing Systems **30** (2017)

[13] Chen, Y.H., Yang, T.J., Emer, J., Sze, V.: Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **9**(2), 292–308 (2019)

[14] Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3), 273–297 (1995)

[15] Curci, S., Mocanu, D.C., Pechenizkiyi, M.: Truly sparse neural networks at scale. arXiv preprint arXiv:2102.01732 (2021)

[16] Defferrard, M., Benzi, K., Vandergheynst, P., Bresson, X.: FMA: A dataset for music analysis. In: 18th International Society for Music Information Retrieval Conference (ISMIR) (2017), `https://arxiv.org/abs/1612.01840`

[17] Defferrard, M., Mohanty, S.P., Carroll, S.F., Salathé, M.: Learning to recognize musical genre from audio. In: The 2018 Web Conference Companion. ACM Press (2018). https://doi.org/10.1145/3184558.3192310, `https://arxiv.org/abs/1803.05337`

[18] Doquet, G., Sebag, M.: Agnostic feature selection. In: Joint european conference on machine learning and knowledge discovery in databases. pp. 343–358. Springer (2019)

[19] Dy, J.G., Brodley, C.E.: Feature selection for unsupervised learning. Journal of machine learning research **5**(Aug), 845–889 (2004)

[20] Evci, U., Gale, T., Menick, J., Castro, P.S., Elsen, E.: Rigging the lottery: Making all tickets winners. In: International Conference on Machine Learning. pp. 2943–2952. PMLR (2020)

[21] Fanty, M., Cole, R.: Spoken letter recognition. Advances in neural information processing systems **3** (1990)

[22] Gu, Q., Li, Z., Han, J.: Generalized fisher score for feature selection. In: 27th Conference on Uncertainty in Artificial Intelligence, UAI 2011. pp. 266–273 (2011)

[23] Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of machine learning research **3**(Mar), 1157–1182 (2003)

[24] Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.A.: Feature extraction: foundations and applications, vol. 207. Springer (2008)

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

[26] He, X., Cai, D., Niyogi, P.: Laplacian score for feature selection. Advances in neural information processing systems **18** (2005)

[27] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research **22**(241), 1–124 (2021)

[28] Hong, C., Sukumaran-Rajam, A., Nisa, I., Singh, K., Sadayappan, P.: Adaptive sparse tiling for sparse matrix multiplication. In: Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming. pp. 300–314 (2019)

[29] Hooker, S.: The hardware lottery. Communications of the ACM **64**(12), 58–65 (2021)

[30] Huang, D.S., Yu, H.J.: Normalized feature vectors: a novel alignment-free sequence comparison method based on the numbers of adjacent amino acids. IEEE/ACM Transactions on Computational Biology and Bioinformatics **10**(2), 457–467 (2013)

[31] Huang, S., Chaudhary, K., Garmire, L.X.: More is better: recent progress in multi-omics data integration methods. Frontiers in genetics **8**, 84 (2017)

[32] Hull, J.J.: A database for handwritten text recognition research. IEEE Transactions on pattern analysis and machine intelligence **16**(5), 550–554 (1994)

[33] Jakulin, A.: Machine learning based on attribute interactions. Ph.D. thesis, Univerza v Ljubljani (2005)

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

[35] Lang, K.: Newsweeder: Learning to filter netnews. In: Machine Learning Proceedings 1995, pp. 331–339. Elsevier (1995)

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

[37] Lemhadri, I., Ruan, F., Tibshirani, R.: Lassonet: Neural networks with feature sparsity. In: International Conference on Artificial Intelligence and Statistics. pp. 10–18. PMLR (2021)

[38] Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J., Liu, H.: Feature selection: A data perspective. ACM computing surveys (CSUR) **50**(6), 1–45 (2017)

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

[40] Li, Y., Chen, C.Y., Wasserman, W.W.: Deep feature selection: theory and application to identify enhancers and promoters. Journal of Computational Biology **23**(5), 322–336 (2016)

[41] Lin, D., Tang, X.: Conditional infomax learning: An integrated framework for feature extraction and fusion. In: European conference on computer vision. pp. 68–82. Springer (2006)

[42] Lindenbaum, O., Shaham, U., Peterfreund, E., Svirsky, J., Casey, N., Kluger, Y.: Differentiable unsupervised feature selection based on a gated laplacian. Advances in Neural Information Processing Systems **34**, 1530–1542 (2021)

[43] Liu, C., Bellec, G., Vogginger, B., Kappel, D., Partzsch, J., Neumärker, F., Höppner, S., Maass, W., Furber, S.B., Legenstein, R., et al.: Memory-efficient deep learning on a spinnaker 2 prototype. Frontiers in neuroscience p. 840 (2018)

[44] Liu, H., Motoda, H.: Feature selection for knowledge discovery and data mining, vol. 454. Springer Science & Business Media (2012)

[45] Liu, J., Ji, S., Ye, J.: Multi-task feature learning via efficient 2, 1-norm minimization. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009. pp. 339–348. AUAI Press (2009)

[46] Liu, S., Chen, T., Atashgahi, Z., Chen, X., Sokar, G., Mocanu, E., Pechenizkiy, M., Wang, Z., Mocanu, D.C.: Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity. In: International Conference on Learning Representations (2022)

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

[48] Liu, S., Mocanu, D.C., Matavalam, A.R.R., Pei, Y., Pechenizkiy, M.: Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. Neural Computing and Applications **33**, 2589–2604 (2020)

[49] Liu, S., Mocanu, D.C., Pei, Y., Pechenizkiy, M.: Selfish sparse rnn training. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 139, pp. 6893–6904. PMLR (18–24 Jul 2021)

[50] Liu, S., Yin, L., Mocanu, D.C., Pechenizkiy, M.: Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In: International Conference on Machine Learning. pp. 6989–7000. PMLR (2021)

[51] Miao, J., Niu, L.: A survey on feature selection. Procedia Computer Science **91**, 919–926 (2016)

[52] Min, F., Hu, Q., Zhu, W.: Feature selection with test cost constraint. International Journal of Approximate Reasoning **55**(1), 167–179 (2014)

[53] Mocanu, D.C., Mocanu, E., Pinto, T., Curci, S., Nguyen, P.H., Gibescu, M., Ernst, D., Vale, Z.A.: Sparse training theory for scalable and efficient agents. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. pp. 34–38 (2021)

[54] Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A.: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nature communications **9**(1), 1–12 (2018)

[55] Nene, S.A., Nayar, S.K., Murase, H., et al.: Columbia object image library (coil-100) (1996)

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

[57] Özdenizci, O., Legenstein, R.: Training adversarially robust sparse networks via bayesian connectivity sampling. In: International Conference on Machine Learning. pp. 8314–8324. PMLR (2021)

[58] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)

[59] Raihan, M.A., Aamodt, T.: Sparse weight activation training. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 15625–15638. Curran Associates, Inc. (2020)

[60] Sheikhpour, R., Sarram, M.A., Gharaghani, S., Chahooki, M.A.Z.: A survey on semi-supervised feature selection methods. Pattern Recognition **64**, 141–158 (2017)

[61] Singh, D., Climente-González, H., Petrovich, M., Kawakami, E., Yamada, M.: Fsnet: Feature selection network on high-dimensional biological data. arXiv preprint arXiv:2001.08322 (2020)

[62] Sokar, G., Mocanu, D.C., Pechenizkiy, M.: Spacenet: Make free space for continual learning. Neurocomputing **439**, 1–11 (2021)

[63] Sokar, G., Mocanu, E., Mocanu, D.C., Pechenizkiy, M., Stone, P.: Dynamic sparse training for deep reinforcement learning. International Joint Conference on Artificial Intelligence (2022)

[64] Spira, A., Beane, J.E., Shah, V., Steiling, K., Liu, G., Schembri, F., Gilman, S., Dumas, Y.M., Calner, P., Sebastiani, P., et al.: Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. Nature medicine **13**(3), 361–366 (2007)

[65] Sun, L., Hui, A.M., Su, Q., Vortmeyer, A., Kotliarov, Y., Pastorino, S., Passaniti, A., Menon, J., Walling, J., Bailey, R., et al.: Neuronal and glioma-derived stem cell factor induces angiogenesis within the brain. Cancer cell **9**(4), 287–300 (2006)

[66] Wang, P., Ji, Y., Hong, C., Lyu, Y., Wang, D., Xie, Y.: Snrram: an efficient sparse neural network computation architecture based on resistive random-access memory. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2018)

[67] Wang, S., Ding, Z., Fu, Y.: Feature selection guided auto-encoder. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 31 (2017)

[68] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: Proceedings of the 26th annual international conference on machine learning. pp. 1113–1120 (2009)

[69] Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)

[70] Yamada, M., Jitkrittum, W., Sigal, L., Xing, E.P., Sugiyama, M.: High-dimensional feature selection by feature-wise kernelized lasso. Neural computation **26**(1), 185–207 (2014)

[71] Yamada, Y., Lindenbaum, O., Negahban, S., Kluger, Y.: Feature selection using stochastic gates. In: International Conference on Machine Learning. pp. 10648–10659. PMLR (2020)

[72] Yang, Y., Shen, H.T., Ma, Z., Huang, Z., Zhou, X.: L2, 1-norm regularized discriminative feature selection for unsupervised. In: Twenty-second international joint conference on artificial intelligence (2011)

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

[74] Zhang, R., Nie, F., Li, X., Wei, X.: Feature selection with multi-view data: A survey. Information Fusion **50**, 158–167 (2019)

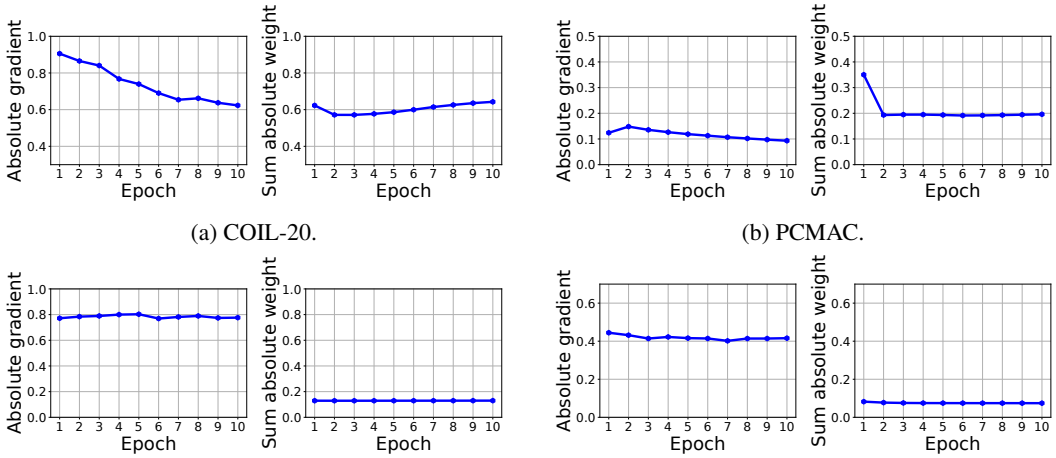
[75] Zhao, Z., Liu, H.: Semi-supervised feature selection via spectral analysis. In: Proceedings of the 2007 SIAM international conference on data mining. pp. 641–646. SIAM (2007)

[76] Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., Li, H.: Learning n: M fine-grained structured sparse neural networks from scratch. In: International Conference on Learning Representations (2020)

[77] Zhu, H., Jin, Y.: Multi-objective evolutionary federated learning. IEEE Transactions on Neural Networks and Learning Systems **31**(4), 1310–1322 (2019)

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] No potential negative societal impacts have been found. See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.3 and Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Tables 2, 5, 6, and Figure 5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Sections 4.3 and 4.4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Sections 4.2 and 4.3.
   (b) Did you mention the license of the assets? [Yes] See Section 4.3.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The code of this work is included in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]



(a) COIL-20.

(b) PCMAC.

(c) SMK.

(d) GLA.

Figure 8: The average values during training of the two components used in the criteria for neuron importance in the input layer: the absolute gradient of the loss with respect to the reconstructed samples and the sum of the absolute weights connected to a neuron.

A Experimental Settings

A.1 Implementation Details

For all datasets, we used standard normalization that scales the features to have zero mean and standard deviation of one. The architecture of the autoencoder consists of one hidden layer with sigmoid activation. A linear activation is used for the output layer. We use a hidden layer of 200 neurons for all datasets. We trained each dataset for 10 epochs using stochastic gradient descent with a momentum of 0.9 and a batch size of 128. We used a learning rate of 0.1 for all datasets except for the datasets with very few samples. We used a learning rate of 0.01 and 0.001 for SMK and GLA, respectively. For sparse-based methods, WAST and QS, we used a sparsity level s of 0.8. The factor of dropped and regrown connections, α, in the two methods is 0.3. We used λ of 0.4 for all datasets of type image, and λ of 0.9 for all other types except the biological data. For SMK and GLA, we used λ of 0.01 and 0.001, respectively. The selection of the value of λ is guided by the magnitude of the gradient of the loss with respect to the reconstructed sample compared to the magnitude of the weights. This is illustrated in Figure 8. We use λ to control the balance between the two components of the neuron importance in Equation 3. For instance, for biological data (SMK and GLA), where the absolute gradient is higher than the average sum of absolute weights connected to a neuron, we use a small value for λ. The other hyperparameters are selected using a random search. For Madelon, we evaluate the performance of $K = 20$. For all other datasets, we test 6 different values $K \in \{25, 50, 75, 100, 150, 200\}$.

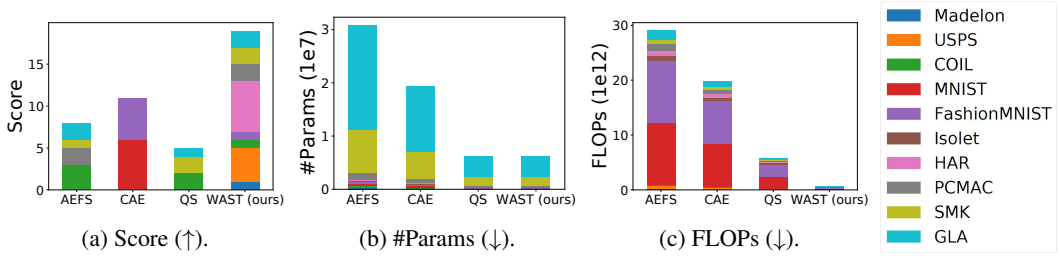
A.2 Evaluation Metrics

Accuracy. To assess how informative are the selected features by each method, we train a support vector machine classifier on the selected features and report the classification accuracy.

Score. To summarize the results for all datasets and various values of the selected features K, we calculate a score for each studied method. The score of a method increases by one when it is the best performer in terms of classification accuracy for a certain dataset and a certain value of K.

Network size (#params). To estimate the memory cost of a network, the network size is calculated by the summation of the number of connections allocated in its layers as follows:

$$\#params = \sum_{l=1}^{L} \left\| W^{(l)} \right\|_0 , \tag{5}$$



(a) Score (↑). (b) #Params (↓). (c) FLOPs (↓).

Figure 9: The performance of different methods on the studied datasets across different dimensions: score (a), memory cost (b), and computational cost (c). The score is accumulated for the 6 studied values for K.

where $W^{(l)}$ is the weights in layer l, $\|.\|_0$ is the standard L_0 norm, and L is the number of layers in the model. For sparse neural networks, $\left\|W^{(l)}\right\|_0$ is controlled by the defined sparsity level for the model.

Floating-point operations (FLOPs). To estimate the computational cost of training a neural network model on a given dataset, we calculate how many FLOPs are performed in the whole course of training. We follow the method described in [20] to calculate the FLOPs. The FLOPs are estimated by the total number of multiplications and additions performed in the forward and backward passes. It is calculated layer by layer in the model and are dependent on the sparsity level of the network.

FLOPs is the typical used metric in the literature to compare the computational cost of a sparse model against its dense counterpart [27]. The main motivation is that current sparse training methods are prototyped using masks over dense weights to simulate sparsity [27]. This is because most deep learning specialized hardware is optimized for dense matrix operations. Therefore, the running time using these prototypes would not reflect the actual gain in memory and speed using a truly sparse network. See Appendix H for further discussion.

Performance. To assess the efficiency of a method across different dimensions (accuracy, memory cost, and computational cost) on various types of datasets with different characteristics, we include a comparison that gives a holistic view of all datasets and the studied values for K. We study 55 cases ((9 datasets $\times$ 6 values for K)+ Madelon dataset). We calculate the total score for all cases and memory and computational costs. Since the memory and computational costs for CAE [5] are dependent on the value of K, we report the average costs across the 6 studied values. The memory and computational efficiency are equivalent to (1− cost). Normalized values, using min-max scaling, are illustrated in Figure 3.

B Additional Experiments

In this appendix, we evaluate the accuracy of the studied unsupervised and supervised methods for various values of the selected features $K \in \{25, 50, 75, 100, 150, 200\}$.

Tables 5 and 6 show the accuracy in each case. Consistent with the observations for K of 50 in Section 4.4, we find that unsupervised neural network-based methods outperform the classical ones in most cases. CAE is the best performer for image datasets with a large number of samples (i.e., MNIST and Fashion MNIST) for all values of K except one. WAST is the best performer in 19 cases, while the second-best performer is the best in 11 cases.

Similarly, the supervised neural network-based methods outperform the classical supervised methods in most cases. Yet, for text and high dimensional datasets, classical methods are competitive with the NN-based ones, outperforming them in 5 values for K on SMK and PCMAC.

Interestingly, WAST outperforms the best supervised performer on the image datasets in 14 cases of 24. On the other types of datasets, supervised methods achieve higher performance in most cases.

Figure 9 summarizes the performance of unsupervised methods across various dimensions. We report the score (i.e., how many times the method is the best performer) on different values of K (Figure 9a), the memory cost estimated by the number of required network parameters (Figure 9b), and the computational cost (Figure 9c). Since the network architecture is dependent on K in CAE [5], we

Table 5: Classification accuracy (%) using unsupervised and supervised feature selection methods for different K selected features. The best performer from the unsupervised methods is in bold font, while the best performer from the supervised methods is in blue.

Dataset			Method/K	25	50	75	100	150	200
USPS	Unsupervised	Classical	lap_score [26]	63.01±0.00	70.54±0.00	86.02±0.00	90.75±0.00	93.87±0.00	95.97±0.00
			MCFS [9]	93.49±0.00	93.33±0.00	**97.15±0.00**	**97.53±0.00**	97.31±0.00	97.26±0.00
			DUFS [42]	90.53±1.97	95.62±0.54	96.37±0.42	96.85±0.39	97.16±0.23	97.49±0.07
		NN-based	AEFS [25]	94.04±0.41	95.86±0.48	96.70±0.17	96.86±0.05	97.00±0.24	96.96±0.22
			CAE [5]	92.50±1.45	95.04±0.59	95.56±0.26	96.10±0.34	95.88±0.23	96.16±0.05
			QS [4]	93.18±0.72	95.88±0.31	96.72±0.32	97.01±0.13	97.41±0.14	97.48±0.05
			WAST (ours)	**94.59±0.74**	**96.69±0.27**	97.02±0.10	97.11±0.11	**97.41±0.13**	**97.65±0.09**
	Supervised	Classical	Fisher_score [22]	81.99±0.00	91.02±0.00	94.35±0.00	96.51±0.00	97.26±0.00	97.53±0.00
			L1_L21 [45]	77.53±0.00	86.99±0.00	91.51±0.00	91.99±0.00	95.54±0.00	96.83±0.00
			CIFE [41]	50.16±0.00	61.29±0.00	67.96±0.00	78.01±0.00	89.57±0.00	96.34±0.00
			ICAP [33]	89.95±0.00	95.22±0.00	95.27±0.00	95.38±0.00	95.75±0.00	96.94±0.00
			RFS [56]	87.37±0.00	95.32±0.00	96.45±0.00	96.72±0.00	97.20±0.00	97.26±0.00
		NN-based	LassoNet [37]	93.56±0.43	95.80±0.12	96.56±0.09	96.98±0.18	97.55±0.05	97.64±0.04
			STG [71]	93.57±0.32	95.78±0.60	96.58±0.27	97.04±0.13	97.37±0.11	97.37±0.05
COIL-20	Unsupervised	Classical	lap_score [26]	60.42±0.00	78.12±0.00	80.90±0.00	82.29±0.00	87.85±0.00	88.19±0.00
			MCFS [9]	91.32±0.00	97.22±0.00	96.53±0.00	98.26±0.00	98.61±0.00	99.65±0.00
			DUFS [42]	92.01±2.91	97.43±1.22	98.06±1.58	98.82±1.53	99.79±0.28	99.65±0.38
		NN-based	AEFS [25]	96.66±1.36	99.48±0.41	99.02±0.56	**99.94±0.12**	**100.00±0.0**	**99.94±0.12**
			CAE [5]	84.08±2.12	94.54±2.92	96.74±1.48	98.34±1.26	97.32±1.05	99.88±0.15
			QS [4]	**97.29±0.71**	99.17±0.42	**99.17±0.47**	98.89±0.60	99.24±0.14	99.10±0.28
			WAST (ours)	94.86±1.39	**99.58±0.14**	99.03± 0.67	98.89±0.92	99.38±0.40	99.51±0.35
	Supervised	Classical	Fisher_score [22]	50.35±0.00	88.89±0.00	93.40±0.00	95.14±0.00	96.18±0.00	97.92±0.00
			L1_L21 [45]	90.97±0.00	92.01±0.00	92.71±0.00	92.71±0.00	98.26±0.00	98.96±0.00
			CIFE [41]	50.69±0.00	59.38±0.00	63.19±0.00	67.71±0.00	71.88±0.00	72.22±0.00
			ICAP [33]	94.44±0.00	99.31±0.00	98.96±0.00	100.0±0.00	100.0±0.00	99.31±0.00
			RFS [56]	34.72±0.00	66.32±0.00	72.22±0.00	78.47±0.00	86.11±0.00	90.28±0.00
		NN-based	LassoNet [37]	91.74±0.94	95.83±1.18	98.89±0.34	99.31±0.00	99.45±0.28	100.0±0.00
			STG [71]	93.40±1.72	97.57±1.7	99.17±0.87	98.89±1.27	98.96±1.39	99.44±0.72
MNIST	Unsupervised	Classical	lap_score [26]	14.14±0.00	23.94±0.00	30.49±0.00	40.15±0.00	50.99±0.00	59.92±0.00
			MCFS [9]	-	-	-	-	-	-
			DUFS [42]	47.23±0.00	62.09±0.00	69.48±0.00	71.27±0.00	87.60±0.00	88.67±0.00
		NN-based	AEFS [25]	87.52±1.47	93.22±1.38	95.78±0.19	96.16±0.50	96.72±0.17	97.14±0.08
			CAE [5]	**91.18±0.68**	**96.20±0.14**	**97.36±0.23**	**97.66±0.08**	**97.90±0.06**	**98.08±0.07**
			QS [4]	87.52±0.74	94.07±0.40	96.00±0.19	96.85±0.20	97.49±0.07	97.88±0.06
			WAST (ours)	88.89±0.88	95.27±0.26	96.76± 0.19	97.27±0.04	97.69±0.09	98.06±0.09
	Supervised	Classical	Fisher_score [22]	78.50±0.00	86.11±0.00	91.14±0.00	94.42±0.00	96.39±0.00	97.51±0.00
			L1_L21 [45]	44.64±0.00	62.26±0.00	75.93±0.00	78.27±0.00	90.56±0.00	90.71±0.00
			CIFE [41]	80.92±0.00	89.30±0.00	92.75±0.00	95.09±0.00	96.76±0.00	97.62±0.00
			ICAP [33]	81.61±0.00	89.03±0.00	92.43±0.00	94.99±0.00	96.44±0.00	97.59±0.00
			RFS [56]	-	-	-	-	-	-
		NN-based	LassoNet [37]	86.30±1.07	94.38±0.12	95.83±0.13	96.57±0.07	97.43±0.09	97.92±0.04
			STG [71]	84.81±0.93	92.53±0.86	95.56±0.33	96.56±0.13	97.53±0.08	97.99±0.06
Fashion MNIST	Unsupervised	Classical	lap_score [26]	18.47±0.00	27.07±0.00	44.16±0.00	53.45±0.00	69.56±0.00	77.32±0.00
			MCFS [9]	-	-	-	-	-	-
			DUFS [42]	45.55±13.86	74.69±1.86	80.35±1.44	83.25±0.67	85.46±0.26	86.40±0.36
		NN-based	AEFS [25]	75.96±1.44	80.88±0.71	82.66±0.90	83.38±0.71	84.30±0.67	85.42±0.49
			CAE [5]	**81.24±0.57**	**84.66±0.16**	**85.74±0.34**	**86.60±0.21**	**86.96±0.22**	87.46±0.17
			QS [4]	77.96±1.11	82.65±0.38	84.51±0.47	85.45±0.46	86.64±0.12	87.04±0.09
			WAST (ours)	73.67±1.59	82.16±0.57	84.42±0.33	85.28±0.07	86.37±0.25	**87.58±0.12**
	Supervised	Classical	Fisher_score [22]	53.10±0.00	67.85±0.00	74.31±0.00	79.59±0.00	83.55±0.00	84.67±0.00
			L1_L21 [45]	63.98±0.00	69.57±0.00	72.1±0.00	72.79±0.00	79.08±0.00	80.53±0.00
			CIFE [41]	63.36±0.00	66.86±0.00	67.66±0.00	69.18±0.00	75.65±0.00	78.78±0.00
			ICAP [33]	50.07±0.00	59.52±0.00	67.22±0.00	77.75±0.00	81.69±0.00	84.53±0.00
			RFS [56]	-	-	-	-	-	-
		NN-based	LassoNet [37]	78.85±0.23	82.63±0.23	84.03±0.09	85.10±0.21	86.02±0.13	86.51±0.11
			STG [71]	76.41±1.59	83.32±0.45	85.04±0.29	86.05±0.28	86.93±0.09	87.35±0.11

report the average memory and computational costs on different K. Datasets with a large number of samples, MNIST and FashionMNIST, cause the highest costs. AEFS requires the highest memory and computational costs. Yet, AEFS performs better than the other dense baseline CAE on various dataset types. QS reduces memory and computational costs substantially. However, it has a lower score than AEFS. WAST achieves the best trade-off across different dimensions. It has the highest score and lowest computational cost.

C Effect of Fast Attention During Training

We studied the performance of unsupervised NN-based methods during the first 10 epochs in Section 5.1. In this appendix, we report the accuracy after performing the 10 epochs.

As illustrated in Table 7, on different dataset types (artificial, image, speech, time series, text, and biological) and different characteristics (high/low dimensional features and few-shot/large data), WAST consistently outperforms other NN-based unsupervised methods.

Table 6: Classification accuracy (%) using unsupervised and supervised feature selection methods for different K selected features. The best performer from the unsupervised methods is indicated in bold font, while the best performer from the supervised methods is indicated in blue color.

Dataset			Method/K	25	50	75	100	150	200
Isolet	Unsupervised	Classical	lap_score [26]	64.81±0.00	75.71±0.00	79.42±0.00	83.85±0.00	86.47±0.00	90.38±0.00
			MCFS [9]	72.37±0.00	81.41±0.00	85.83±0.00	88.46±0.00	91.35±0.00	91.79±0.00
			DUFS [42]	**73.92±2.57**	**85.62±2.53**	**89.72±2.31**	**92.04±1.50**	**93.60±1.12**	**94.37±0.43**
		NN-based	AEFS [25]	73.76±1.13	80.94±2.02	88.04±1.59	89.80±0.72	91.66±0.73	90.76±1.66
			CAE [5]	60.32±5.55	78.90±1.24	82.00±2.17	84.52±1.58	86.66±1.24	87.24±1.11
			QS [4]	62.77±0.56	74.62±2.12	82.17±1.40	87.31±1.28	92.33±0.23	93.46±0.34
			WAST (ours)	70.90±1.99	85.33±1.39	87.60±1.03	88.44±0.76	90.23±0.19	91.46±0.37
	Supervised	Classical	Fisher_score [22]	68.72±0.00	75.64±0.00	83.53±0.00	86.60±0.00	89.42±0.00	93.78±0.00
			L1_L21 [45]	48.72±0.00	55.90±0.00	56.67±0.00	66.92±0.00	72.31±0.00	73.33±0.00
			CIFE [41]	56.03±0.00	59.81±0.00	74.29±0.00	81.22±0.00	85.71±0.00	87.95±0.00
			ICAP [33]	67.05±0.00	75.06±0.00	79.68±0.00	82.82±0.00	89.29±0.00	90.26±0.00
			RFS [56]	66.54±0.00	77.31±0.00	85.06±0.00	87.76±0.00	92.50±0.00	94.87±0.00
		NN-based	LassoNet [37]	76.78±0.32	85.70±0.38	90.49±0.08	93.23±0.55	95.15±0.06	95.67±0.06
			STG [71]	77.37±4.54	89.38± 1.19	92.43±0.57	93.92±0.16	94.85±0.09	95.74±0.17
HAR	Unsupervised	Classical	lap_score [26]	80.83±0.00	82.80±0.00	83.78±0.00	84.66±0.00	89.48±0.00	92.77±0.00
			MCFS [9]	60.10±0.00	80.29±0.00	84.39±0.00	83.78±0.00	91.08±0.00	91.04±0.00
			DUFS [42]	73.96±5.37	86.90±1.06	87.91±1.90	88.81±1.23	92.18±0.70	93.34±0.96
		NN-based	AEFS [25]	82.28±6.01	89.54±0.44	90.76±0.49	91.56±0.39	91.52±0.96	93.30±1.29
			CAE [5]	85.70±2.97	86.26±2.41	89.60±0.62	90.16±1.19	90.34±0.60	91.26±0.92
			QS [4]	78.62±1.15	89.68±0.38	89.73±1.77	89.96±0.58	91.68±0.76	92.90±0.46
			WAST (ours)	**86.37±0.51**	**91.20±0.16**	**91.99±0.20**	**93.97±0.18**	**95.15±0.25**	**94.96±0.49**
	Supervised	Classical	Fisher_score [22]	81.27±0.00	83.68±0.00	88.84±0.00	89.89±0.00	93.08±0.00	92.30±0.00
			L1_L21 [45]	79.4±0.00	81.30±0.00	83.31±0.00	84.90±0.00	91.92±0.00	93.76±0.00
			CIFE [41]	80.22±0.00	84.15±0.00	84.83±0.00	85.34±0.00	85.92±0.00	85.92±0.00
			ICAP [33]	84.46±0.00	88.70±0.00	89.24±0.00	92.06±0.00	93.38±0.00	93.32±0.00
			RFS [56]	84.22±0.00	88.23±0.00	88.53±0.00	89.92±0.00	91.31±0.00	92.67±0.00
		NN-based	LassoNet [37]	92.69±0.26	93.93±0.15	94.62±0.15	95.04±0.24	95.49±0.18	95.45±0.10
			STG [71]	86.44±1.45	91.75±0.59	93.00±0.14	93.75±0.37	94.12±0.37	94.24±0.09
PCMAC	Unsupervised	Classical	lap_score [26]	49.61±0.00	49.87±0.00	50.64±0.00	55.78±0.00	55.53±0.00	58.10±0.00
			MCFS [9]	51.67±0.00	53.47±0.00	53.98±0.00	62.72±0.00	**75.06±0.00**	**72.75±0.00**
			DUFS [42]	54.96±2.02	57.79±3.18	59.90±3.28	60.52±1.63	65.04±2.59	67.92±1.28
		NN-based	AEFS [25]	54.14±1.92	60.40±2.37	**63.40±6.19**	**65.10±1.97**	65.24±3.11	67.48±2.89
			CAE [5]	54.30±0.00	55.08±0.00	56.24±1.86	57.12±2.47	62.28±4.28	66.12±3.04
			QS [4]	54.65±3.25	55.78±3.25	59.79±1.28	61.65±2.53	63.86±2.67	67.61±4.19
			WAST (ours)	**58.77±2.48**	**60.51±2.53**	62.37±1.56	61.59±2.71	65.09±3.50	67.51±2.31
	Supervised	Classical	Fisher_score [22]	81.75±0.00	86.38±0.00	85.60±0.00	84.58±0.00	84.58±0.00	84.83±0.00
			L1_L21 [45]	54.50±0.00	53.98±0.00	55.27±0.00	56.81±0.00	59.64±0.00	60.41±0.00
			CIFE [41]	77.12±0.00	75.84±0.00	74.81±0.00	72.49±0.00	72.49±0.00	75.58±0.00
			ICAP [33]	82.78±0.00	87.66±0.00	87.92±0.00	87.40±0.00	87.92±0.00	88.43±0.00
			RFS [56]	73.78±0.00	67.61±0.00	72.24±0.00	70.95±0.00	74.29±0.00	73.52±0.00
		NN-based	LassoNet [37]	86.79±0.31	86.53±1.25	85.30±1.19	85.35±0.86	84.63±0.44	84.83±0.92
			STG [71]	54.4±2.36	56.04±1.90	59.07±2.29	62.11±3.31	65.09±2.88	68.84±2.26
SMK	Unsupervised	Classical	lap_score [26]	**84.21±0.00**	81.58±0.00	84.21±0.00	81.58±0.00	84.21±0.00	84.21±0.00
			MCFS [9]	65.79±0.00	78.95±0.00	78.95±0.00	81.58±0.00	78.95±0.00	78.95±0.00
			DUFS [42]	79.47±6.32	81.05±3.07	81.05±3.87	83.16±3.57	83.16±3.16	82.63±2.68
		NN-based	AEFS [25]	72.64±3.57	79.48±3.07	85.80±3.57	**85.76±3.59**	81.58±1.68	82.62±2.12
			CAE [5]	77.90±3.17	78.94±2.37	79.48±2.60	83.12±3.57	84.20±1.64	85.78±2.12
			QS [4]	81.05±5.37	81.58±3.72	81.58±2.88	81.05±4.21	**85.26±2.11**	**86.32±1.97**
			WAST (ours)	76.32±4.99	**84.74±1.05**	**86.84±2.35**	85.26±2.68	84.74±1.05	85.79±1.29
	Supervised	Classical	Fisher_score [22]	68.42±0.00	73.68±0.00	76.32±0.00	78.95±0.00	78.95±0.00	78.95±0.00
			L1_L21 [45]	78.95±0.00	84.21±0.00	89.47±0.00	84.21±0.00	81.58±0.00	81.58±0.00
			CIFE [41]	81.58±0.00	81.58±0.00	76.32±0.00	81.58±0.00	81.58±0.00	78.95±0.00
			ICAP [33]	78.95±0.00	73.68±0.00	71.05±0.00	76.32±0.00	71.05±0.00	76.32±0.00
			RFS [56]	78.95±0.00	76.32±0.00	76.32±0.00	71.05±0.00	71.05±0.00	71.05±0.00
		NN-based	LassoNet [37]	75.79±6.74	77.37±3.57	80.00±1.29	81.58±2.88	80.00±4.88	81.05±1.05
			STG [71]	76.32±2.88	81.05±5.37	80.53±1.29	81.58±4.07	81.05±3.07	84.21±0.00
GLA	Unsupervised	Classical	lap_score [26]	**72.22±0.00**	66.67±0.00	66.67±0.00	69.44±0.00	69.44±0.00	72.22±0.00
			MCFS [9]	69.44±0.00	75.00±0.00	69.44±0.00	72.22±0.00	66.67±0.00	77.78±0.00
			DUFS [42]	66.66±5.56	70.83±1.39	73.61±1.39	72.22±0.00	69.44±2.77	70.83±1.39
		NN-based	AEFS [25]	66.68±3.51	67.76±6.21	71.10±3.76	71.12±4.51	**74.44±2.10**	**73.32±2.86**
			CAE [5]	65.00±6.25	70.56±4.50	71.66±3.23	72.20±1.77	73.88±4.19	72.22±3.51
			QS [4]	67.78±5.72	68.89±4.78	72.22±3.04	**73.33±3.83**	72.22±1.76	71.67±2.08
			WAST (ours)	66.67±4.65	**75.56±4.08**	**74.44±3.24**	71.11±3.77	71.11±1.36	69.44±3.04
	Supervised	Classical	Fisher_score [22]	58.33±0.00	63.89±0.00	66.67±0.00	66.67±0.00	63.89±0.00	61.11±0.00
			L1_L21 [45]	69.44±0.00	69.44±0.00	72.22±0.00	75.00±0.00	72.22±0.00	72.22±0.00
			CIFE [41]	61.11±0.00	58.33±0.00	58.33±0.00	58.33±0.00	72.22±0.00	72.22±0.00
			ICAP [33]	69.44±0.00	72.22±0.00	72.22±0.00	69.44±0.00	69.44±0.00	72.22±0.00
			RFS [56]	-	-	-	-	-	-
		NN-based	LassoNet [37]	75.56±3.24	76.67±2.22	75.56±1.11	75.00±0.00	75.56±1.11	75.00±0.00
			STG [71]	68.33±8.71	71.11±2.83	73.33±2.84	73.33±2.84	73.33±3.77	72.22±2.49

Table 7: Classification accuracy (%) (↑) using unsupervised NN-based feature selection methods. 50 features are selected for all datasets except Madelon, where 20 features are used. All methods are trained for 10 epochs.

Method	Madelon	USPS	COIL-20	MNIST	FashionMNIST
AEFS [25]	53.50±1.17	95.46±0.39	96.18±2.15	93.83±0.63	79.45±3.26
CAE [5]	61.20±6.61	94.94±1.37	96.20±1.10	91.60±1.33	80.54±0.46
QS [4]	52.73±2.28	95.55±0.46	97.15±0.13	94.01±0.37	79.29±0.40
WAST (ours)	**83.27±0.63**	**96.69±0.27**	**99.58±0.14**	**95.27±0.26**	**82.16±0.57**

Method	Isolet	HAR	PCMAC	SMK	GLA
AEFS [25]	81.98±3.17	86.43±3.40	57.23±2.57	78.42±3.87	70.00±2.72
CAE [5]	78.30±4.57	89.02±1.54	61.16±3.45	81.04±4.52	73.34±3.32
QS [4]	84.24±1.56	86.61±2.83	58.09±2.10	83.15±2.10	71.11±2.83
WAST (ours)	**85.33±1.39**	**91.20±0.16**	**60.51±2.53**	**84.74±1.05**	**75.56±4.08**

Table 8: Classification accuracy (%) (↑) using unsupervised NN-based feature selection methods on the FMA dataset.

Method/K	25	50	75
AEFS [25]	38.50±1.11	41.20±1.48	42.74±1.32
CAE [5]	37.06±1.17	39.00±0.84	39.90±1.32
QS [4]	41.50±2.29	44.65±1.10	**45.57±0.37**
WAST (ours)	**42.13±2.51**	**44.98±1.10**	45.45±0.48

D Performance Evaluation on Music Dataset

In this appendix, we perform additional experiments on a recent music dataset, Free Music Archive (FMA) [16; 17]. FMA includes a collection of 106574 audio tracks with artist and album information categorized into 161 genres. We use FMA-small, which consists of 6400 training samples and 800 testing samples with 140 pre-extracted features. We focus on the genre classification task that categorizes the data into eight classes. We use the same setting defined in Appendix A. For WAST, we use λ of 0.4. We evaluate the classification accuracy of unsupervised NN-based methods using $K = \{25, 50, 75\}$.

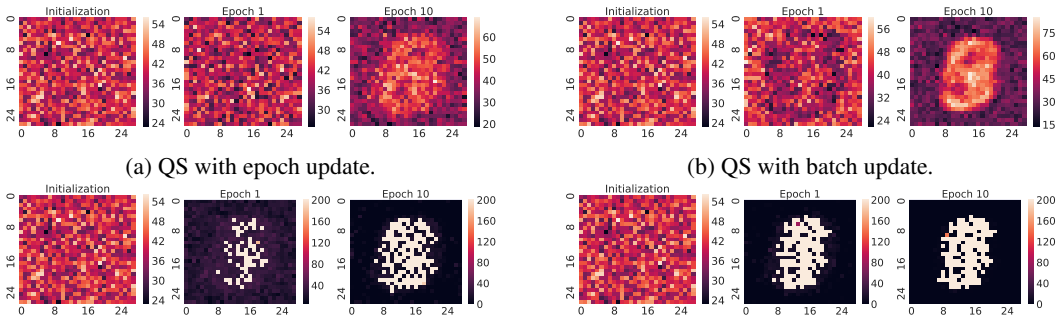
The results are reported in Table 8. Consistent with our results on other benchmarks, WAST outperforms other unsupervised NN-based methods in two cases out of three, with a marginal performance difference in the third case. The performance gain is accompanied by a reduction of the number of epochs by 90%.

E Effect of the Frequency of Topology Update

In this section, we analyze the effect of the schedule used to update the sparse topology. In WAST, the topology is updated after each weight update using a batch of the data. We named this schedule "batch update". We compare this schedule with the one used in the QS baseline, in which the sparse topology is updated after each training epoch (i.e., one pass on the whole data). We denote this schedule as "epoch update".

Table 9: Effect of update schedule of the sparse topology in WAST. The accuracy (%) is reported using K of 20 and 50 on Madelon and others, respectively.

Update schedule	Madelon	USPS	HAR	PCMAC	SMK
Epoch update	**83.37±0.62**	96.17±0.30	90.55±0.52	57.02±3.53	84.21±1.29
Batch update	83.27±0.63	**96.69±0.27**	**91.20±0.16**	**60.51±2.53**	**84.74±1.05**



(a) QS with epoch update.

(b) QS with batch update.

(c) WAST with epoch update.

(d) WAST with batch update.

Figure 10: Effect of the update schedule of the sparse topology in the QS baseline and WAST. The sparse topology is updated after each training epoch (left) or after batch update of the weights (right). Updating the topology more frequently increases the speed of detecting the informative features.

Table 9 shows the accuracy on datasets from different types. Updating the topology more frequently by the batch update schedule improves the performance in most of the cases and helps in finding the informative features quicker (Appendix F). The accuracy increases by $0.52 - 3.5\%$ in the studied cases.

F Visualizing the Effect of the Frequency of Topology Update

In this appendix, we visualize the effect of different update schedules of the sparse topology. We analyzed two schedules: "epoch update" and "batch update" (Appendix E). We performed this analysis on MNIST.

Figure 10 illustrates the effect of different topological update schedules in WAST and QS. The less frequent update of the topology decreases the speed of identifying the informative features. After 1 epoch, the distribution of connections by QS is still almost uniform across different input neurons (Figure 10a). On the other hand, WAST starts to identify the important neurons (Figure 10c). Updating the topology after each batch update increases the speed of identification in the random exploration setting (Figure 10b) and enables very fast detection in WAST after 1 epoch (Figure 10d).

G Robustness to Noisy Environments

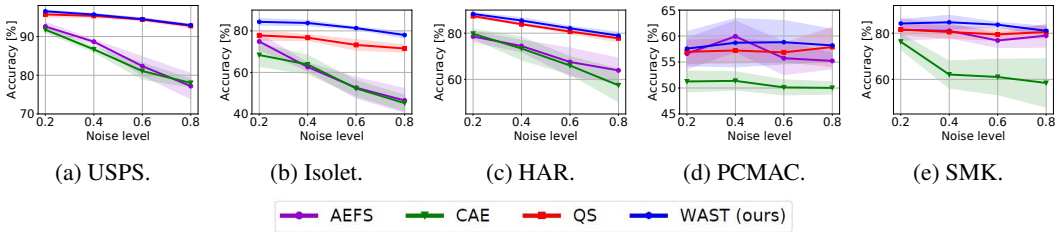
Our experiments on the Madelon dataset illustrate the robustness of WAST to noisy features. In this appendix, we study the robustness of the unsupervised NN-based methods on noisy samples. To this end, we created noisy versions of the datasets by adding Gaussian noise with zero mean to the training data. To assess the robustness of different noise levels, we evaluated 4 different values of standard deviation $\{0.2, 0.4, 0.6, 0.8\}$. We performed this analysis on datasets of different types. We use the same experimental setting described in Section 4.3 and Appendix A.

Figure 11 illustrates the classification accuracy using $K = 50$. We observe that across different dataset types, methods that are based on sparse models (i.e., WAST and QS) are more robust to noise. As expected, the accuracy decreases when the level of noise increases in all methods. Yet, sparse-based methods have the least decrease. Moreover, WAST outperforms QS in all cases.

H Advance in Sparsity

Most of the efforts in previous years are devoted to the algorithmic side of sparse training. The aim is to achieve the same or higher performance than the dense models with highly sparse counterparts, as discussed in Section 2.2.

Existing sparse training methods in the literature simulate sparsity using masks over dense weights since most deep learning specialized hardware is optimized for dense matrix operations. Yet, recently, research on the software and hardware that support sparsity has received growing attention. NVIDIA



(a) USPS. (b) Isolet. (c) HAR. (d) PCMAC. (e) SMK.

Figure 11: The performance of different methods on noisy datasets where Gaussian noise is added. We assess the performance on different noise levels by varying the standard deviation of the Gaussian noise. The test accuracy is reported using K of 50.

released NVIDIA A100, which supports a 50% fixed sparsity level [76], and many other efforts on the hardware side are proposed [66; 3; 13; 43]. On the software side, libraries that support truly sparse implementations have been started for supervised learning [48; 15]. With a joint community effort in the algorithmic, software, and hardware directions, we would be able to actually provide faster, memory-efficient, and energy-efficient deep neural networks. Further discussion can be found in [29; 53].