Advancing Efficiency in Neural Networks through Sparsity and Feature Selection

Zahra Atashgahi

Advancing Efficiency in Neural Networks through Sparsity and Feature Selection

Zahra Atashgahi

Advancing Efficiency in Neural Networks through Sparsity and Feature Selection

Zahra Atashgahi

This dissertation has been approved by::

Promotors:
prof. dr. ir. R.N.J. Veldhuis
prof. dr. M. Pechenizkiy

Co-promotors:
dr. ir. D.C. Mocanu

UNIVERSITY
OF TWENTE.

Keywords:	sparse neural networks, dynamic sparse training, feature selection, efficiency, cost-effective neural networks.
ISBN (print):	978-90-365-5981-2
ISBN (digital):	978-90-365-5982-9
DOI:	`https://doi.org/10.3990/1.9789036559829`
Cover image:	Generated using DALL.E 3
Cover design:	Ipskamp printing
Printed by:	Ipskamp printing

Advancing Efficiency in Neural Networks through Sparsity and Feature Selection

Dissertation

To obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. ir. A. Veldkamp,
on account of the decision of the Doctorate Board
to be publicly defended
on Tuesday 30 April 2024 at 16.45

By

Zahra Atashgahi

born on the 7th of January, 1995
in Neyshaboor, Iran

Graduation Committee:

Chair / secretary:	prof. dr. J.N. Kok
Promotors:	prof. dr. ir. R.N.J. Veldhuis Universiteit Twente, EEMCS, Data management & Biometrics
	prof. dr. M. Pechenizkiy Eindhoven University of Technology (TU/e)
Co-promotor:	dr. ir. D.C. Mocanu University of Luxembourg
Committee Members:	prof. dr. C. Seifert University of Marburg
	dr. M.E. Taylor University of Alberta
	prof. dr. C. de Campos Eindhoven University of Technology (TU/e)
	prof. dr. C. Brune Universiteit Twente, EEMCS, Mathematics of Imaging & AI
	prof. dr. A.J. Schmidt-Hieber Universiteit Twente, EEMCS, Mathematics of Operations Research

To all the brave women in Iran
To Women, Life, Freedom

Summary

Deep neural networks (DNNs) have attracted considerable attention over the last several years due to their promising results in various applications. Nevertheless, their extensive model size and over-parameterization have brought to the forefront a significant challenge—escalating computational costs. Furthermore, these challenges are exacerbated when dealing with high-dimensional data, as the complexity and resource requirements of DNNs increase significantly. Consequently, the utilization of deep learning models proves to be ill-suited for scenarios characterized by constrained computational resources and limited battery life, incurring substantial training and inference costs, both in terms of memory and computational resources.

Sparse neural networks (SNNs) have emerged as a prominent approach toward addressing the over-parameterization inherent in DNNs, thus, mitigating associated costs. By keeping only the most important connections of a DNN, they achieve a comparable result to their dense counterpart network but with significantly fewer parameters. However, most current solutions to reduce computation costs using SNNs mainly gain inference efficiency, while being resource-intensive during training. Furthermore, these solutions predominantly center their efforts on a restricted set of application domains, particularly within the realms of vision and language tasks.

This Ph.D. research aims to address these challenges by introducing **C**ost-**e**ffective **A**rtificial **N**eural **N**etworks (CeANNs) designed to achieve a targeted performance across diverse complex machine learning tasks while demanding minimal computational, memory, and energy resources during both network training and inference. Our study on CeANNs includes two primary perspectives:

model and data efficiency. In essence, we leverage the potential of SNNs to reduce the model parameters and data dimensionality, thereby facilitating efficient training and deployment of artificial neural networks. This work results in the development of artificial neural networks that are more practical and accessible for real-world applications, with a key emphasis on cost-effectiveness. Within this thesis, we delve into our developed methodologies aimed at advancing efficiency. Our contributions can be summarized as follows:

Part I. Advancing Training and Inference Efficiency of DNNs through Sparsity. This part of the thesis focuses on enhancing the model efficiency of DNNs through sparsity. The inherent high computational cost associated with DNNs, primarily stemming from their large, over-parameterized layers, highlights the need for computationally-aware design in both model architecture and training methods. Within Part I of this thesis, we leverage sparsity to address this challenge, with a specific focus on achieving a targeted performance in extremely sparse neural networks and efficient time series analysis with DNNs. We propose two algorithms to tackle these issues: a dynamic sparse training (DST) algorithm for learning in extremely sparse neural networks (Chapter 2) and a methodology for obtaining SNNs for time series prediction (Chapter 3). In essence, our goal is to enhance the training and inference efficiency of DNNs through sparsity while focusing on addressing specific challenges in underexplored application domains, particularly in tabular and time series data analysis.

Part II. Leveraging Feature Selection for Efficient Model Development. In the pursuit of cost-effective artificial neural networks, it is crucial to address the challenges associated with high-dimensional input data due to its potential to hinder scalability and introduce issues such as the curse of dimensionality and over-fitting. One promising avenue to tackle these challenges is feature selection, a technique designed to identify the most relevant and informative attributes of a dataset. However, existing feature selection methods are mostly computationally expensive, especially when dealing with high-dimensional datasets or those with a substantial sample size. To address this issue, in the second part of the thesis, we propose for the first time to exploit SNNs to perform efficient feature selection. We present our two proposed feature selection methods, one for unsupervised feature selection (Chapter 4) and another for supervised feature selection (Chapter 5). These methods are specifically designed to offer effective solutions to the challenges of high dimensionality while maintaining computational efficiency. As we show in Chapter 4, by using less than 10% of

the parameters of the dense network, our proposed method achieves the highest ranking-based score in terms of finding qualitative features among the state-of-the-art feature selection methods. The combination of feature selection and neural networks offers a powerful strategy, enhancing the training process and performing dimensionality reduction, thereby advancing the overall efficiency of model development.

In conclusion, this research focuses on the development of cost-effective artificial neural networks that deliver targeted performance while minimizing computational, memory, and energy resources. The research explores CeANNs from two perspectives: model efficiency and data efficiency. The first part of the thesis addresses model efficiency through sparsity, proposing algorithms for efficient training and inference of DNNs for various data types. The second part of the thesis leverages SNNs to efficiently select an informative subset of attributes from high-dimensional input data. By considering both model and data efficiency, the aim is to develop CeANNs that are practical and accessible for real-world applications. In Chapter 6, we present the preliminary impact and the limitations of the work and potential directions for future research in the field. We hope that this Ph.D. thesis will pave the way to designing cost-effective artificial neural networks.

Samenvatting

Diepe neurale netwerken (DNN's) hebben de afgelopen jaren aanzienlijke aandacht getrokken vanwege hun veelbelovende resultaten in verschillende toepassingen. Desondanks heeft hun uitgebreide modelgrootte en over-parameterisatie een aanzienlijke uitdaging naar voren gebracht: toenemende computationele kosten. Bovendien worden deze uitdagingen verergerd wanneer omgegaan wordt met hoog-dimensionale gegevens, aangezien de complexiteit en resourcevereisten van DNN's aanzienlijk toenemen. Als gevolg is het gebruik van deep learning modellen niet geschikt voor scenario's die gekenmerkt worden door beperkte computationele resources en beperkte batterijlevensduur, wat aanzienlijke trainings- en inferentiekosten met zich meebrengt, zowel in termen van geheugen als computationele resources.

Schrale neurale netwerken (SNN's) zijn naar voren gekomen als een prominente benadering om de over-parameterisatie die inherent is aan DNN's aan te pakken, waardoor de bijbehorende kosten worden verminderd. Door alleen de belangrijkste verbindingen van een DNN te behouden, bereiken ze een vergelijkbaar resultaat met hun dichte tegenhanger netwerk maar met aanzienlijk minder parameters. Echter, de meeste huidige oplossingen om computationele kosten te verminderen met SNN's behalen voornamelijk efficiëntie tijdens de inferentie, terwijl ze resource-intensief zijn tijdens de training. Bovendien concentreren deze oplossingen hun inspanningen voornamelijk op een beperkte set van toepassingsdomeinen, vooral binnen de gebieden van visie en taaltaken.

Dit PhD-onderzoek streeft ernaar deze uitdagingen aan te pakken door **K**osten-**e**ffectieve Kunstmatige **N**eurale **N**etwerken (KeNN's) te introduceren die ontworpen zijn om een gerichte prestatie te bereiken over diverse complexe

machine learning taken, terwijl ze minimale computationele, geheugen- en energiebronnen eisen tijdens zowel netwerktraining als -inferentie. Onze studie naar KeNN's omvat twee primaire perspectieven: model- en data-efficiëntie. In wezen benutten we het potentieel van SNN's om de modelparameters en data dimensionaliteit te verminderen, waardoor efficiënte training en implementatie van kunstmatige neurale netwerken wordt gefaciliteerd. Dit werk resulteert in de ontwikkeling van kunstmatige neurale netwerken die praktischer en toegankelijker zijn voor echte toepassingen, met een sleutel nadruk op kosten-effectiviteit. Binnen deze thesis verdiepen we ons in onze ontwikkelde methodologieën gericht op het bevorderen van efficiëntie. Onze bijdragen kunnen als volgt worden samengevat:

Deel I. Het bevorderen van trainings- en inferentie-efficiëntie van DNN's door middel van spaarzaamheid. Dit deel van de thesis richt zich op het verbeteren van de model efficiëntie van DNN's door spaarzaamheid. De inherent hoge computationele kosten geassocieerd met DNN's, voornamelijk voortkomend uit hun grote, over-geparameteriseerde lagen, benadrukken de noodzaak voor computationeel bewust ontwerp in zowel modelarchitectuur als trainingsmethoden. Binnen Deel I van deze thesis benutten we spaarzaamheid om deze uitdaging aan te gaan, met een specifieke focus op het bereiken van een gerichte prestatie in extreem schrale neurale netwerken en efficiënte tijdreeksanalyse met DNN's. We stellen twee algoritmen voor om deze problemen aan te pakken: een dynamisch schraal trainingsalgoritme (DST) voor leren in extreem schrale neurale netwerken en een methodologie voor het verkrijgen van SNN's voor tijdreeksvoorspelling.

Deel II. Het benutten van kenmerkselectie voor efficiënte modelontwikkeling. In de zoektocht naar kosteneffectieve kunstmatige neurale netwerken is het cruciaal om de uitdagingen aan te pakken die geassocieerd zijn met hoog-dimensionale invoergegevens, vanwege het potentieel om schaalbaarheid te hinderen en problemen zoals de vloek van dimensionaliteit en overfitting te introduceren. Een veelbelovende manier om deze uitdagingen aan te gaan is kenmerkselectie, een techniek ontworpen om de meest relevante en informatieve attributen van een dataset te identificeren. Echter, bestaande methoden voor kenmerkselectie zijn meestal computationeel duur, vooral wanneer ze te maken hebben met hoog-dimensionale datasets of die met een aanzienlijke steekproefgrootte. Om dit probleem aan te pakken, stellen we in het tweede deel van de thesis voor het eerst voor om SNN's te gebruiken voor efficiënte kenmerkselectie.

We presenteren onze twee voorgestelde methoden voor kenmerkselectie, één voor onbegeleide kenmerkselectie en een ander voor begeleide kenmerkselectie. Deze methoden zijn specifiek ontworpen om effectieve oplossingen te bieden voor de uitdagingen van hoge dimensionaliteit, terwijl de computationele efficiëntie behouden blijft. Zoals we laten zien, door minder dan 10% van de parameters van het dichte netwerk te gebruiken, bereikt onze voorgestelde methode de hoogste rangschikkingsgebaseerde score in termen van het vinden van kwalitatieve kenmerken onder de state-of-the-art methoden voor kenmerkselectie. De combinatie van kenmerkselectie en neurale netwerken biedt een krachtige strategie, die het trainingsproces verbetert en dimensionaliteitsreductie uitvoert, waardoor de algehele efficiëntie van modelontwikkeling wordt bevorderd.

Concluderend richt dit onderzoek zich op de ontwikkeling van kosteneffectieve kunstmatige neurale netwerken die gerichte prestaties leveren terwijl ze de computationele, geheugen- en energiebronnen minimaliseren. Het onderzoek verkent KeNN's vanuit twee perspectieven: model efficiëntie en data efficiëntie. Het eerste deel van de thesis richt zich op model efficiëntie door spaarzaamheid, en stelt algoritmen voor voor efficiënte training en inferentie van DNN's voor verschillende datatypes. Het tweede deel van de thesis benut SNN's om efficiënt een informatieve subset van attributen te selecteren uit hoog-dimensionale invoergegevens. Door zowel model- als data efficiëntie te overwegen, is het doel om KeNN's te ontwikkelen die praktisch en toegankelijk zijn voor echte toepassingen. In het slotkapittel presenteren we de voorlopige impact en de beperkingen van het werk en potentiële richtingen voor toekomstig onderzoek in het veld. We hopen dat deze PhD-thesis de weg zal banen voor het ontwerpen van kosteneffectieve kunstmatige neurale netwerken.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Networks (DNNs) have demonstrated remarkable performance in various machine learning tasks [HZRS16, KSH12, RDGF16, VSP+17, BMR+20, HDY+12, EKN+17]. This success can be attributed to the extremely large over-parameterized models which improve the generalization on unseen samples [NLB+19, AZLS19, BGMSS18, DZPS19]. However, this over-parameterization leads to the ever-increasing cost of computational and memory resources during the training and deployment of these models [HABN+21]. The high computational and memory demands can be severely worsened when DNNs are applied to high-dimensional and/or noisy data, leading to issues such as slow convergence during training, the curse of dimensionality, and over-fitting [GBC16]. Consequently, training and inference of DNNs on low-resource devices, e.g., an edge device with limited computational, memory, and energy resources, might not be economically viable.

Furthermore, such an increase in computations stemming from training and deployment of deep learning models can lead to a critical rise in the energy consumption in data centers and, consequently, results in extremely high carbon emissions and adverse environmental effects [SGM20]. Notably, discussions in [SGM20] reveal that CO_2 emissions from the training and deployment of some Natural Language Processing (NLP) models can exceed the average annual carbon footprint of a human by more than seven times.

This Chapter is partly based on: Zahra Atashgahi, *Cost-effective Artificial Neural Networks,* **International Joint Conference on Artificial Intelligence (IJCAI)**, Doctoral Consortium, 2023.

According to the High-Level Expert Group on Artificial Intelligence, ensuring environmental well-being is a crucial aspect of building trustworthy AI systems [Gro20, BLN+23]. It is essential to evaluate the development, deployment, and operation of AI systems to guarantee their adherence to environmentally friendly practices. Therefore, we believe it is necessary to develop *Cost-effective Artificial Neural Networks* which are defined as follows:

Definition 1.1. **C**ost-**e**ffective **A**rtificial **N**eural **N**etworks (**CeANNs**) are artificial neural networks that are designed to deliver a targeted performance while minimizing the required computational power, memory, and energy consumption, for their development and deployment.

CeANNs can be explored from two main perspectives: model and data.

- **Model.** As discussed earlier, training and deploying DNNs suffer from high computational cost which is attributed to their large over-parameterized models. Developing CeANNs necessitates a rethinking of the architecture and training methodologies applied in current deep learning models. The central challenge lies in reducing the computational, memory, and energy requirements while preserving or even improving predictive performance. Model compression through sparsity is a widely used technique to overcome the challenges raised by the over-parametrization of neural networks. Sparsification techniques can result in a 10-100 times decrease in model size, leading to associated theoretical improvements in computational, storage, and energy efficiency, all without significantly comprising accuracy [HABN+21]. In this thesis, we employ sparsity in model architecture during training and inference to develop CeANNs that are efficient during both development and deployment.

- **Data.** Another major bottleneck in decreasing the costs associated with training neural networks is noisy and high-dimensional input data. Input data to ANNs can be high-dimensional. High dimensionality poses several challenges, including the curse of dimensionality and the risk of overfitting, which can compromise the performance and generalization ability of neural networks. Furthermore, the large volume of input dimensions can hinder the scalability and computational efficiency of ANNs. Moreover, the input to neural networks might contain noisy or irrelevant features, which can impede the training process by slowing it down or causing it to diverge from the optimal solution. Addressing the challenges posed by noisy and high-dimensional input data is paramount to ensuring the cost-effectiveness of

artificial neural networks. Feature selection is a popular technique that can facilitate the training process by selecting the most informative features from the input data, leading to improved training and more efficient and economically viable neural networks. In this thesis, we leverage feature selection and sparsity to efficiently address the high costs associated with data.

By reducing the costs raised by the model and/or data without compromising the predictive performance, we can achieve CeANNs that enable efficient training and deployment of ANNs, making them more practical and accessible for real-world applications.

This thesis evaluates CeANNs' training and inference efficiency through various metrics. Chapter 4 presents a genuinely sparse implementation of the proposed method, assessing efficiency based on running time, memory usage, and energy consumption. In contrast, other chapters utilize a binary mask to simulate sparsity, with computational efficiency measured by floating-point operations (FLOPs) and the number of parameters. Using FLOPs and parameter counts to estimate the efficiency benefits of sparse neural networks over their dense counterparts is a well-established method [EGM+20, SMM+21]. The parameter count reflects the model's size, influencing both memory demands and computational complexity. FLOPs provide an implementation-independent measure of an algorithm's time complexity. Moreover, since current deep learning hardware lacks optimization for sparse matrix operations, many approaches to achieving sparse neural networks merely mimic sparsity through a binary weight mask. As a result, the actual running time of these methods may not accurately indicate their efficiency. Additionally, the pursuit of true sparse implementations for sparse neural networks is an active area of research [Hoo21, HABN+21]. Given the theoretical nature of our work, we have deferred this aspect of engineering research to future endeavors. Consequently, we also rely on parameters and FLOPs counts to assess efficiency.

1.1 Advancing Model Efficiency through Sparsity

Sparse neural networks (SNNs) have gained significant attention as an effective solution to the issue of over-parameterization in DNNs. They are achieved by setting a subset of model parameters to zero and as a result, reducing the complexity of the model [HABN+21]. SNNs can be defined as below:

Definition 1.2. A sparse neural network is characterized by a connectivity graph $G(\mathcal{V}, \mathcal{E}')$, where $\mathcal{V}$ denotes the set of vertices (nodes or neurons), $\mathcal{E}$ represents the set of edges (connections or parameters) in the equivalent dense neural network, and $\mathcal{E}'$ is a subset of edges from the dense network ($\mathcal{E}' \subset \mathcal{E}$). The cardinality of $\mathcal{E}'$ is significantly smaller than that of $\mathcal{E}$, denoted as $|\mathcal{E}'| \ll |\mathcal{E}|$, resulting in a sparse connectivity among components.

By keeping only the most important parameters of a DNN, SNNs achieve performance on par with their dense counterparts while having a much smaller size. SNNs not only mitigate the issue of model over-parameterization but also have the potential to accelerate model training, improve inference efficiency, and facilitate the deployment of neural networks in resource-constrained environments.

Sparsity in neural networks can be classified into two main categories: unstructured and structured. Unstructured sparsity includes pruning arbitrary indices in the weights based on specific criteria, necessitating the storage of zero-element indices. This can lead to limited acceleration on typical GPUs due to the overhead associated with indexing. In contrast, structured sparsity targets contiguous blocks of weights, such as neurons, filters, and channels, rather than individual elements. This reduces the indexing overhead and improves the computational efficiency during execution. While structured sparsity has demonstrated computational efficiency on standard hardware, unstructured sparsity has been shown to achieve higher accuracy at higher sparsity levels [MHP+17] and improve performance in various aspects beyond efficiency such as adversarial robustness [CZpw+22], data efficiency [VTCZ+22], and out-of-distribution generalization/detection [SL22, NSvKS23, LW23], interpretability [CYM+23]. Therefore, in this thesis, we leverage the potential of unstructured sparsity to develop CeANNs. Furthermore, we design CeANNs that benefit from real speedup through a truly sparse implementation.

Sparse neural networks can be achieved through two primary methods: pruning dense neural networks, often referred to as *dense-to-sparse* approach, or training SNNs directly from scratch, known as *sparse-to-sparse* approach [HABN+21, MMP+21].

1.1.1 Dense-to-sparse

Dense-to-sparse (a.k.a. pruning) methods prune a dense neural network to the desired sparsity level. Pruning techniques can be categorized based on the phase at which pruning occurs: *post-training* [LDS90, HPTD15], *during-training* [ZG17],

and *before-training* [LAT19] pruning.

- **Post-training.** Post-training pruning is the most widely used approach for obtaining SNNs. Seminal works include Optimal Brain Damage (OBD) [LDS90] and Optimal Brain Surgeon (OBS) [HS93] that exploit hessian matrix information to assess the connection importance and prune unnecessary connections from a dense network in order to reduce the computational complexity. [HPTD15] refined the idea for deep learning models and demonstrated that it is feasible to decrease the storage and computation of DNNs by an order of magnitude without without sacrificing accuracy. This was achieved by iterative pruning and retraining in a three-step process: first, training the network to recognize important connections; second, pruning the unimportant ones; and lastly, fine-tuning the weights of the remaining connections. Post-training pruning methods might require one or several rounds of pruning and retraining [HPTD15, FC19]. Various metrics are used to assess connection importance in this process, such as Hessian matrix information [LDS90, HS93], magnitude [HPTD15, FC19], gradient [LW19], Taylor expansion [MTK+17, MMT+19], and low-rank decomposition [WGFZ19, LGM+20]. While post-training pruning achieves on-par performance with dense neural networks, it suffers from high computational costs due to the dense training of the network. It can be even more costly than training a dense neural network due to multiple rounds of pruning and retraining.

- **During-training.** To address the high computational costs of post-training approaches, during-training methods gradually reduce the number of parameters throughout training until they reach the desired sparsity level before the training completion [ZG17, LWK18, LCC+21]. A standard during-training pruning is Gradual Magnitude Pruning (GMP) [ZG17] which gradually drops unimportant weights based on the magnitude during the training process. These methods can maintain close performance to the dense network while being more efficient during training.

- **Before-training.** Before-training pruning methods find a well-performing sparse sub-network at initialization. The idea of pruning at initialization was first introduced by [LAT19] proposing Single-shot Network Pruning (SNIP). Before-training algorithms prune the network to the desired sparsity level before the training starts based on various connection saliency scores, such as sensitivity to training loss [LAT19, dJSB+21], gradient preservation [WZG19], synaptic strength [TKYG20], and meta gradi-

ents [ATZ+22], or based on the input-output path [PD21]. While the entire training process occurs in a sparse manner, these methods require at least a few iterations of dense training. Therefore, they fall into the category of dense-to-sparse training. It is worth noting that before-training pruning approaches typically lag behind the performance of their dense counterparts.

The post-training approach often proves to be the most computationally demanding among dense-to-sparse methods, as it necessitates one or multiple rounds of dense training. Consequently, the resource requirements decrease for during-training and before training, impacting the potential accuracy achievable. Thus, the choice among these three methods depends on the available training resources, enabling practitioners to make informed decisions tailored to their computational constraints.

While unstructured pruning holds the potential to achieve dense model performance, its reliance on dense matrices during training necessitates a careful consideration of the performance-resource tradeoff, contingent upon available resources. In environments with limited computational resources, especially where accommodating the dense model proves unfeasible on existing hardware, applying these models becomes impractical. Therefore, it becomes imperative to assess the computational capabilities and constraints prior to opting for dense-to-sparse pruning, ensuring a careful allocation of resources for optimal performance.

1.1.2 Sparse-to-sparse

Sparse-to-sparse training refers to a class of methods used to train sparse neural networks from scratch, aiming to achieve computational efficiency during both training and inference. These methods train a sparse network throughout the entire training process. The sparse-to-sparse approach is more efficient than the dense-to-sparse methods as it never uses dense matrices before or throughout training. The sparse connectivity of the network can be fixed (a.k.a. static sparse training [MMN+16]) or dynamically optimized (a.k.a. dynamic sparse training [MMS+18]) during training. One main drawback of static sparse training is that the sparse connectivity is initialized randomly before training and cannot be adapted to the data automatically during training. Therefore, its performance lags behind that of the dense network. Dynamic sparse training addresses this by optimizing the sparse connectivity during training.

Dynamic Sparse Training (DST) updates the sparse topology iteratively by

dropping a fraction of unimportant weights and growing an equal number of weights. DST term was first introduced in [MW19] and it stems from SET [MMS+18], quickly followed by DeepR [BKML18] and NeST [DYJ19]. While the drop criterion is often the magnitude of the weights, the growing criteria can be of different types [HABN+21], including random-based [MMS+18], gradient-based [EGM+20, JPR+20], and locality-based weight regrowth. DST methods have shown promising results, often achieving comparable or even superior performance to dense networks while requiring significantly fewer training and inference FLOPs.

However, despite the promise of sparse neural networks, there are several issues with the existing methods. One notable concern is the weight regrowth mechanism in DST. Current methods predominantly rely on random, which can slow down the convergence, or gradient-based criteria which necessitates dense gradient computations at regular intervals during training. This added computational load can substantially increase the training overhead. Additionally, the performance of DST methods tends to suffer in extremely sparse regions. Furthermore, the existing techniques for achieving sparse neural networks have been primarily tailored to applications in vision and natural language processing, leaving a notable gap when it comes to applying sparse neural networks to other data types, such as time series or tabular data. These issues highlight the need for further research and development in the field of sparse neural networks to overcome these challenges.

1.2 Leveraging Feature Selection for Designing Cost-effective Models

Feature selection techniques can help reduce computational overhead, enhance predictive accuracy, and offer deeper insights into the data by identifying the most relevant and informative features within a given dataset [CS14].

There are three main categories of feature selection methods: filter, wrapper, and embedded methods. Filter methods rely on data characteristics and employ ranking criteria, such as correlation [GE03], and mutual information [Bat94], to evaluate features independently of the learning task. While these methods are fast and efficient, they are prone to selecting redundant features or features that are irrelevant to the target learning task. In contrast, wrapper methods rely on the predictive performance of a learning function and employ search strategies [ZNLW19], such as tree structures [KJ97] and evolutionary algorithms [LS+96]

to identify feature subsets that maximize predictive performance. Although wrapper methods often yield better performance than filter approaches, they are computationally expensive due to the large search space of features, particularly on high-dimensional data. Embedded methods overcome the limitations of filter and wrapper approaches by integrating feature selection into the training process to find the relevant subset of features for the task at hand [Bat94, PLD05, GWBV02, SL97]. The embedded model selects the features that contribute most to the accuracy of the training model [DA22]. They exploit the feature interactions with the learning algorithm, thus, finding better features than filter methods. Additionally, as embedded methods do not require iterative evaluation of selected features, they prove to be more computationally efficient than wrapper methods [LCW+18].

Neural network-based feature selection has gained significant attention in recent years, both in supervised [LFLN18, LRAT21, YLNK20, WC20] and unsupervised [BAZ19, HWZ+18, CS15, DS19] settings. These methods leverage the advantages of neural networks in capturing non-linear dependencies and performing well on large datasets. However, many existing neural network-based feature selection methods suffer from over-parameterization, resulting in high computational costs, especially for high-dimensional datasets. Consequently, when dealing with extremely high-dimensional data, it may not be feasible to directly employ dense neural networks due to the extremely large model size and the lack of sufficient computational and memory resources to fit and train the model.

1.3 Research Questions

In this Ph.D. research, we aim to leverage the power of SNNs and feature selection in developing Cost-effective Artificial Neural Networks (CeANNs) (Definition 1.1). The research questions of this thesis are as follows:

(Q1) **How can we design ANNs that can deliver a targeted performance, while minimizing the utilization of computational power and memory resources during both the training and inference stages?**

DST is an approach to gain efficiency in both the training and inference phases of ANNs. Despite the good performance of the existing DST algorithms from low to high sparsity levels, the effectiveness of SNNs in extreme sparsity regimes, when trained from scratch, has remained unsatisfactory. To enable

model training and deployment in resource-constrained environments, it is crucial to facilitate learning of DST in extreme sparsity regimes.

(Q2) **Can we use sparse neural networks to learn from time series efficiently?**

Despite the extensive body of literature on sparse neural networks designed in the domain of vision and language, exploring sparsity in models for time series analysis has remained understudied. Developing computationally efficient models to perform time series analysis has become increasingly crucial due to the continuous growth in the number of large time series collections and the demand to forecast millions of time series [THA+18, HLW16, RCM+12].

(Q3) **How can we address the challenges imposed by high-dimensional data efficiently using SNNs?**

Feature selection is a popular approach to address the challenges raised by high-dimensional data. However, most existing methods suffer from high computational costs when applied to high-dimensional datasets. Developing computationally efficient feature selection methods for high-dimensional data is of great importance to the community.

All in one, we seek to answer *How we can reduce the training and deploying costs associated with today's deep learning models without compromising performance?*

1.4 Thesis Contributions and Outline

In this section, we outline the structure of the thesis and provide a brief overview of the main chapters and their respective contents. By presenting a clear roadmap for the thesis, this outline aims to guide the reader through the subsequent sections.

This thesis consists of two main parts, each focusing on an aspect of designing CeANNs, including model and data efficiency, respectively:

Part I Advancing Training and Inference Efficiency of DNNs through Sparsity (See Section 1.4.1)

Part II Leveraging Feature Selection for Designing Cost-effective Models (See Section 1.4.2)

In the following, we shed light on the content of each part which are the contributions of this thesis.

1.4.1 Part I. Advancing Training and Inference Efficiency of DNNs through Sparsity

As discussed in Section 1.1, sparsity is one of the keys to achieving efficiency in deep neural networks. However, the pursuit of model efficiency using SNNs presents several challenges, which prompted the formulation of research questions 1 and 2 discussed in Section 1.3. To improve the performance of DST in extremely sparsity regimes, inspired by the Hebbian learning theory, we propose a novel DST algorithm in Chapter 2. We introduce a new pruning algorithm to prune transformers for time series forecasting in Chapter 3. In the following, we provide a brief overview of each of the chapters in the first part of the thesis:

Chapter 2

In this chapter, we introduce a dynamic sparse training algorithm named "Cosine similarity-based and Random Topology Exploration (CTRE)" [APL+22]. CTRE draws inspiration from the Hebbian learning theory to adapt the connectivity of a sparse neural network within the Dynamic Sparse Training (DST) framework. By leveraging cosine similarity as a metric for connection importance, CTRE evolves sparse network topologies, adding the most important connections without relying on dense gradient information. Through extensive experiments across diverse datasets, including tabular, image, and text data, we demonstrate that CTRE outperforms several state-of-the-art sparse training algorithms, particularly in highly sparse network scenarios. CTRE addresses the pressing challenges of computational efficiency and resource constraints, offering a more environmentally sustainable and economically viable solution for neural network development and deployment. Additionally, the promising results of applying CTRE to Multi-Layer Perceptrons (MLPs) for tabular data, which is shown to constitute a significant computational workload in data centers, pave the way for environmentally friendly ANNs. Our findings highlight the effectiveness and applicability of the CTRE algorithm in diverse domains, ultimately advancing sparse neural network training.

Chapter 3

Despite the promising results of CTRE in learning from diverse data types such as tabular, image, and text, its primary emphasis has been on classification using MLPs. Yet, the potential of sparse neural networks in the context of time series analysis, particularly in forecasting, remains unexplored. On the other hand, sparse network literature predominantly covers vision and natural language processing, neglecting pruning in time series analysis, despite the growing need for computationally efficient models to handle large time series datasets and millions of forecasts. Therefore, in Chapter 3, we aim to learn from time series data efficiently [APVM23]. Particularly, we focus on decreasing the computational and memory costs of training and deploying transformers for time series forecasting. We first show that determining the optimal sparsity level when learning from time series data is challenging due to variations in the loss-sparsity trade-offs across datasets. Then, we propose "Pruning with Adaptive Sparsity Level" (PALS), a novel approach that dynamically balances loss and sparsity without requiring a predefined sparsity level. PALS integrates dynamic sparse training with pruning and introduces the "Expand" mechanism, offering a novel perspective in the field of sparse neural networks. Using these mechanisms and loss heuristics, PALS automatically finds a decent trade-off between loss and sparsity in one round of training. By performing experiments on six benchmark datasets and five SOTA transformer variants for time series forecasting, we show that PALS reduces the model's size (reducing 65% parameters on average) and computation (63% FLOPs reduction on average) substantially while maintaining comparable performance to the dense counterpart in terms of prediction loss.

1.4.2 Part II. Leveraging Feature Selection for Designing Cost-effective Models

The second part of the thesis seeks to answer Q3 in Section 1.3, which focuses on alleviating the burden of high computational costs and memory requirements imposed by the rise of high-dimensional data. Feature selection, which identifies the most relevant and informative attributes of a dataset, not only addresses the computational costs caused by the high dimensionality of data but also can help to increase interpretability and improve generalization. However, most existing feature selection methods are computationally expensive. To address this research question, we present two developed feature selection methods. In chapter 4, we introduce QuickSelection, which performs feature selection using sparse neural network characteristics to perform unsupervised feature

selection. In Chapter 5, we propose another feature selection method that integrates feature selection into the training of a sparse neural network to perform supervised feature selection.

Chapter 4

In Chapter 4, we introduce the first method to use the characteristics of sparse neural networks to perform energy-efficient feature selection, named QuickSelection [ASvdL+22]. QuickSelection introduces the strength of the neuron in SNNs as a criterion to measure the feature importance. It incorporates sparsely connected denoising autoencoders trained with the DST framework to model data distribution efficiently, ultimately reducing memory usage and training time. Using an SNN to perform feature selection, QuickSelection achieves the best trade-off between accuracy and computational efficiency when compared with several baseline methods. Experimental results on benchmark datasets reveal QuickSelection's superior performance in terms of classification and clustering accuracy, running time, and memory usage, making it a more energy-efficient choice compared to other neural network-based feature selection methods. QuickSelection marks a big leap forward in our journey to make data processing more efficient and environmentally friendly, especially as datasets keep getting larger and more complex.

Chapter 5

Although QuickSelection provides valuable insights into the characteristics of sparse neural networks and how they can be leveraged to rank the input features, the training process is not tailored to perform feature selection. The proposed strength metric may not accurately rank features in a very high-dimensional feature space, such as those with 40,000 features. In Chapter 5, we introduce NeuroFS to enhance feature selection in high-dimensional data. NeuroFS integrates feature selection into the training of sparse neural networks, effectively reducing the search space during the feature selection process [AZK+23]. The core innovation of NeuroFS lies in its dynamic neuron pruning and regrowing approach within the input layer of a sparse neural network during training. This unique process allows NeuroFS to adaptively identify and retain informative features while discarding uninformative ones. Furthermore, because NeuroFS operates in a supervised manner, it can leverage loss signals to select features based on the underlying classification task. NeuroFS's ability to perform feature selection efficiently is evaluated across a range of real-world datasets. Results

demonstrate that NeuroFS outperforms other state-of-the-art supervised feature selection models, making it a valuable tool for data analysis, particularly in high-dimensional feature space and in scenarios where resource constraints are a concern.

Chapter 6

Chapter 6 brings together the key findings and insights gleaned from this Ph.D. research, shedding light on the broader implications and significance of our work. We also highlight the limitations of our study, paving the way for future investigations in this field. Finally, the chapter concludes by outlining promising avenues for further research.

1.5 How to Read This Thesis

This thesis is designed with self-contained chapters, offering readers the flexibility to explore topics of interest in any order. While reading sequentially provides a gradual introduction to the concepts, individual chapters can be explored independently, as they cover specific aspects of the topic. This structure accommodates various reading preferences and allows readers to delve into the material according to their interests or research needs.

Chapter 2

A Brain-inspired Algorithm for Training Highly Sparse Neural Networks

In this chapter, we aim to obtain sparse neural networks that can be trained and deployed efficiently in terms of computational and memory costs and perform decently in the high-sparsity regime. Inspired by the evolution of the biological brain and the Hebbian learning theory, we propose a new dynamic sparse training approach that evolves sparse neural networks according to the behavior of neurons in the network. Concretely, by exploiting the cosine similarity metric to measure the importance of the connections, our proposed method, Cosine similarity-based and Random Topology Exploration (CTRE), evolves the topology of sparse neural networks by adding the most important connections to the network without calculating the dense gradient in the backward path. We carry out different experiments on eight datasets, including tabular, image, and text datasets, and demonstrate that our proposed method outperforms several state-of-the-art sparse training algorithms in extremely sparse neural networks by a large gap. The implementation code is available at `https://github.com/zahraatashgahi/CTRE`.

This Chapter is integrally based on: Zahra Atashgahi, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy, *A brain-inspired algorithm for training highly sparse neural networks,* **Machine Learning**, **ECML-PKDD 2022 journal track**, Vol. 111, No. 12, pp. 4411–4452, 2022.

2.1 Introduction

Dense artificial neural networks are a commonly used machine learning technique that has a wide range of application domains, such as speech recognition [GMH13], image processing [LH15, MWHN18], and natural language processing (NLP) [BMR+20]. It has been shown in [HNA+17] that the performance of deep neural networks scales with model size and dataset size, and generalization benefits from over-parameterization [NLB+19]. However, the ever-increasing size of deep neural networks has given rise to major challenges, including high computational cost during training and inference and a high memory requirement [ZZL+20]. Such an increase in the number of computations can lead to a critical rise in the energy consumption in data centers and, consequently, a deteriorative effect on the environment [YXJ+18]. However, a trustworthy AI system should function in the most environmentally friendly way possible during development, and deployment [Gro20]. In addition, such gigantic computational costs will lead to a situation where on-device training and inference of neural network models on low-resource devices, e.g., an edge device with limited computational resources and battery life, might not be economically viable [ZZL+20].

Sparse neural networks have been proposed as an effective solution to address these challenges [HABN+21, MMP+21]. By using sparsely connected layers instead of fully connected ones, sparse neural networks have reached a competitive performance to their dense equivalent networks in various applications [FC19, ASvdL+22], while having much fewer parameters. It has been shown that biological brains, especially the human brain, enjoy sparse connections among neurons [Fri08].

Most existing solutions to obtain sparse neural networks focus on inference efficiency in order to reduce the storage requirement of deploying the network and the prediction time of test instances. This class of methods, named *dense-to-sparse* training, starts by training a dense neural network followed by a pruning phase that aims to remove unimportant weight from the network. As categorized in [MMP+21], in dense-to-sparse training, the pruning phase can be done after training [LDS90, FC19, HPTD15], during training [LWK18], or one-shot before training [LAT19]. However, starting from a dense network leads to a memory requirement of fitting a dense network on the device and the computational resources for at least a few iterations of training the dense model. Therefore, training sparse neural networks using dense-to-sparse methods might be infeasible on low-resource devices due to the energy and computational resource constraints.

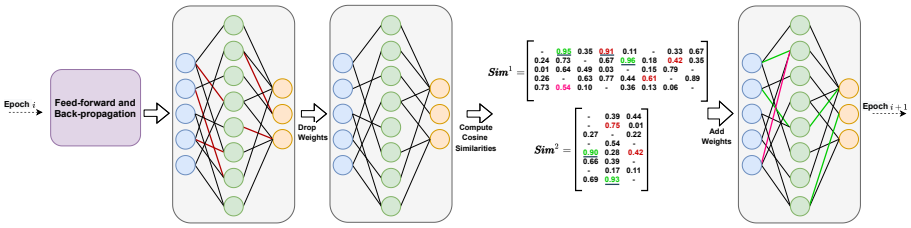


Figure 2.1: Schematic of the proposed approach (CTRE$_{\text{sim}}$). At each epoch, after feed-forward and back-propagation, a fraction ζ of the weights with the smallest magnitude is dropped (red connections). Then, the similarity matrices $\boldsymbol{Sim}^1$ and $\boldsymbol{Sim}^2$ are computed using Equation 2.2 to find the most important connections to add to the network; however, we do not consider the similarity of the existing connections (empty entries). Finally, the weights corresponding to the highest similarity values in the similarity matrices (underlined values) that have not been dropped in the weight removal step are added to the network (underlined green values), the same amount as removed previously. If a connection with high similarity has been dropped in the weight removal step (underlined red value), a random connection will be inserted instead (pink connection).

With the emergence of the *sparse training* concept in [MMN+16], there has been a growing interest in training sparse neural networks that are sparse from scratch. This sparse connectivity could be fixed during training (known as *static sparse training* [MMN+16, MMP+21, KR19]), or might dynamically change, by removing and regrowing weights (known as *dynamic sparse training* [MMS+18, BKML18]). By optimizing the topology along with the weights during the training, dynamic sparse training algorithms outperform the static ones [MMS+18]. As discussed in [MMS+18], weight removal in dynamic sparse training algorithms is similar to synapse shrinkage in the human brain during sleep, where weak synapses shrink and strong ones remain unchanged.

While most dynamic sparse training methods use magnitude as a pruning criterion, weight regrowing approaches are of different types, including random [MMS+18, MW19] and gradient-based regrowth [EGM+20, JPR+20]. As shown in [LYMP21], the random addition of weights could lead to a low training speed, and the performance of sparse training is highly correlated with the total number of parameters explored during training. To speed up convergence, the gradient information of non-existing connections can be used to add the most important

connections to the network [DZ19]. However, computing the gradient of all non-existing connections in a sparse neural network can be computationally demanding. Consequently, increasing the size of the network might escalate the high computational cost into a bottleneck in the sparse training of networks on low-resource devices. In addition, in Section 2.4.2, we demonstrate that some gradient-based sparse training algorithms might fail in a highly sparse neural network.

In this chapter, to address some of these challenges, we introduce a biologically plausible algorithm for obtaining a sparse neural network. By taking inspiration from the Hebbian learning theory, which states "neurons that fire together, wire together" [Heb05], we introduce a new weight addition policy in the context of sparse training algorithms. Our proposed method, "Cosine similarity-based and Random Topology Exploration (CTRE)"[1], takes advantage of both the similarity of neurons as an importance metric for connections and the random search simultaneously (CTRE_{sim}, Figure 2.1) or sequentially (CTRE_{seq}) to find a performant subnetwork. In short, our contributions are as follows:

- We propose a novel and biologically plausible algorithm for training sparse neural networks, which has a limited number of parameters during training. Our proposed algorithm, CTRE, exploits both the similarity of neurons and the random search to find a performant sparse topology.
- We introduce the Hebbian learning theory in the training of the sparse neural networks. Using the cosine similarity of each pair of neurons in two consecutive layers, we determine the most important connections at each epoch during sparse training of the network. We discuss in detail why this approach is an extension to the Hebbian learning theory in Section 2.3.2.
- Our proposed algorithms outperform state-of-the-art sparse training algorithms in highly sparse neural networks.

While deep learning models have shown great success in vision and NLP tasks, these models have not been fully explored in the domain of tabular data [PMB19]. However, designing deep models that are capable of processing tabular data is of great interest to researchers as it paves the way to building multi-modal pipelines for problems [GRKB21]. This work mainly focuses on Multi-Layer Perceptrons (MLPs), which are commonly used for tabular and biological data. Despite the simple structure of MLPs and having only a few

[1] To enhance readability, CTRE is utilized in place of CRTE (the sequential order of the initial letters).

hyperparameters to adjust, they have shown good performance in classification tasks [GS21, THK+21]. Furthermore, in [JYP+17], the authors investigated that, despite the massive attention on CNN architectures, they utilize only 5% of the neural network workload of TPUs in Google data centers, while MLPs constitute 61% of the total workload. Therefore, it is crucial to develop an efficient algorithm that can accelerate MLPs and be resource-efficient during training and inference. To pursue this goal, in this research, we aim to design sparse MLPs with a limited number of parameters during training and inference. To demonstrate the validity of our proposed algorithm, in addition to evaluating the methods on tabular and text datasets, we compare the methods on image datasets such as MNIST, Fashion-MNIST, and CIFAR-10/100 datasets which are commonly used as benchmarks in previous studies.

2.2 Background

In this section, we provide an overview of the background information regarding this research.

2.2.1 Sparse Neural Networks

Methods to obtain and train sparse neural networks can be stratified into two major categories: dense-to-sparse and sparse-to-sparse. In the following, we shed light on each of these two approaches.

Dense-to-sparse

Dense-to-sparse methods to obtain sparse neural networks start training from a dense model and then prune the unimportant connections. They can be divided into three major subcategories: (1) *Pruning after training*: Most existing dense-to-sparse methods start with a trained dense network and iteratively (one or several iterations) prune and retrain the network to reach desired sparsity level. Seminal works were performed in the 1990s in [LDS90, HS93], where authors use hessian matrix information to prune a trained dense network. More recently, in [HPTD15, FC19], authors use magnitude to remove unimportant connections. Other metrics, such as gradient [LW19], Taylor expansion [MTK+17, MMT+19], and low-rank decomposition [WGFZ19, LGM+20], have been also employed to prune the network. While being effective techniques in terms of the performance of the obtained sparse network, these methods suffer from high

computational costs during training. (2) *Pruning during training*: To decrease the computational cost, this group of methods performs pruning during training [GEH19, JZR+19, KRS+20]. Various criteria can be used for pruning, such as magnitude [GYC16, ZG17], L_0 regularization [LWK18, SSM20], group Lasso regularization [WWW+16], and variational dropout [MAV17]. (3) *Pruning before training*: The first study to apply pruning prior to training was done by [LAT19] in [LAT19], that used connection sensitivity to remove weights. Later works have followed the same approach by pruning the network before training using different approaches, such as gradient norm after pruning [WZG19], connection sensitivity after pruning [dJSB+21], and Synaptic Flow [TKYG20].

Sparse-to-sparse

To lower the computational cost of dense-to-sparse methods, sparse-to-sparse training algorithms (also known as sparse training) use a sparse network from scratch with sparse connectivity, which might be static (static sparse training [MMN+16, KR19]) or dynamic (dynamic sparse training (DST) [MMS+18, BKML18]). By allowing the topology to be optimized along with the weights, sparse neural networks trained with DST have reached a comparable performance to the equivalent dense networks or even outperform them.

DST methods can be divided into two main categories based on the weight addition policy:

1. *Random regrowth*: Sparse Evolutionary Training (SET) [MMS+18] is one of the earliest works that starts with a sparse neural network and performs magnitude pruning and random weight regrowing at each epoch to update the topology. In [MW19], the authors proposed the idea of parameter reallocation automatically across layers during sparse training in CNNs. Many works have further studied sparse training concept recently [GEN+18, ASvdL+22, LCC+21, LvdLY+20, LMPP21, LYMP21].

2. *Gradient information*: A group of works tried to exploit gradient information to speed up the training process in DST [RA20]. [DZ19] used the momentum of the non-existing connections as a criterion to grow weights instead of random addition in the SET algorithm; While being effective in terms of accuracy, this method requires computing gradients and updating the momentum for all non-existing parameters. The Rigged Lottery (RigL) [EGM+20] addressed the high computational cost by using infrequent gradient information. However, it still requires the computational

cost for computing the periodic dense gradients. [JPR+20] tried to further improve RigL by using the gradient for only a subset of non-existing weights. In [DYJ19], authors exploit gradient information in the search for a performant sub-network and discuss that gradient-based weight addition is biologically plausible.

2.2.2 Hebbian Learning Theory

The Hebbian learning rule was proposed in 1949 by [Heb05] as the learning rule for neurons inspired by biological systems. It describes how the neurons' activations influence the connections among them. The classical Hebb's rule indicates "neurons that fire together, wire together". This can be formulated as $\Delta w_{ij} = \eta p_i q_j$, where Δw_{ij} is the change in synaptic weight w_{ij} between two neurons p_i (presynaptic) and q_j (postsynaptic) in two consecutive layers, and η is the learning rate.

While some previous works have adapted Hebb's rule to a few machine learning tasks, [SB16, LGM17, SHK23], it has not been vastly investigated in many others, particularly in the sparse neural networks. By adapting Hebb's rule to artificial neural networks, we can obtain powerful models that might be close to the function of structures found in neural systems of various species [KMST15]. In [ABGM14], authors have incorporated the Hebbian learning theory to train a newly introduced neural network. In [SWT16], the Hebbian learning concept has been used to sparsify the neural networks for face recognition; they drop the connections between the weakly correlated neurons. In [DYJ19], authors proposed a gradient-based algorithm for obtaining a sparse neural network; they discuss the gradient-based connection growth policy is mathematically close to the Hebbian learning theory. In this work, by taking inspiration from the Hebbian learning theory, we aim to introduce a new sparse training algorithm for obtaining sparse neural networks.

2.2.3 Cosine Similarity

In most machine learning problems, the Euclidean distance is a common tool to measure the distance due to its simplicity. However, the Euclidean distance is highly sensitive to the vectors' magnitude [XZL15]. Cosine similarity is another metric that addresses this issue; it measures the similarity of the shapes of two vectors as the cosine of the angle between them. In other words, it determines whether the two vectors are pointing in the same direction or not [HKP+12].

Due to its simplicity and efficiency, cosine similarity is a widely used metric in machine learning and pattern recognition fields [XZL15]. It often measures the document similarity in natural language processing tasks [SGGAP14, LH13]. Cosine Similarity has proven to be an effective tool in neural networks. In [LZX+18], to bound the pre-activations in a multi-layer neural network that might disturb the generalization, authors have proposed to use cosine similarity instead of the dot product and showed that it reaches a better performance than the simple dot product. In [NB10], authors have used this metric to improve face verification using deep learning.

2.3 Proposed Method

In this section, we first formulate the problem. Secondly, we demonstrate that cosine similarity can be leveraged as a tool for determining the importance of weights in neural networks and how it relates to the Hebbian learning theory. Finally, we present two new sparse training algorithms using cosine similarity-based connection importance.

2.3.1 Problem Definition

Given a set of training samples $\mathbb{X}$ and target output $\boldsymbol{y}$, a dense neural network is trained to minimize $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})$, where m is the number of training samples, L is the loss function, f is a neural network parameterized by $\boldsymbol{\theta}$, $f(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$ is the predicted output for input $\boldsymbol{x}^{(i)}$, and $\boldsymbol{y}^{(i)}$ is the true label. $\boldsymbol{\theta} \in \mathbb{R}^N$ is consisted of parameters of each layer $l \in \{1,2,...,H\}$ of the network as $\boldsymbol{\theta}^l \in \mathbb{R}^{N^l}$, where $N^l = n^{l-1} \times n^l$ is the number of parameters of layer l, n^l is number of neurons at layer l, and the total number of parameters of the dense network is N. A sparse neural network, however, uses only a subset of $\boldsymbol{\theta}^l$, and discards s^l fraction of parameters of each layer $\boldsymbol{\theta}^l$ (their weight values are equal to zero); s^l is referred to as the *sparsity* of layer l. The overall sparsity of the network is $S = 1 - D$, where $D = \frac{\sum_{l=1}^{H}(1-s^l)N^l}{N}$ is the overall density of the network. We aim to obtain a sparse neural network with sparsity level of S and parameters $\boldsymbol{\theta}$. We aim to train this network to minimize the loss on the training set as follows:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\mathbb{R}^N,\ \|\boldsymbol{\theta}\|_0 = D\times N}{\arg\min} \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)}), \tag{2.1}$$

where $\|\boldsymbol{\theta}\|_0$ is the total number of non-zero connections of the network which is determined by the density level.

Network Structure. The architecture we consider is a Multi-layer Perceptron (MLP) with H layers. Initially, sparse connections between two consecutive layers are initialized with an Erdős–Rényi random graph. Each connection in this graph exists with a probability of $P(\theta_i^l) = \frac{\varepsilon(n^{l-1}+n^l)}{n^{l-1}n^l}$, $i \in \{1,2,...,N^l\}$, where $\varepsilon \in \mathbb{R}^+$ denotes the hyperparameter that controls the sparsity level. The lower the value of ε is, the sparser the network would be. In other words, by increasing ε, the probability of $P(\theta_i^l)$ would be higher which results in more connections and a denser network. Each existing connection is initialized with a small value from a normal distribution.

2.3.2 Cosine Similarity to Determine Connections Importance

In this work, we use cosine similarity as a metric to derive the importance of non-existing connections and evolve the topology of a sparse neural network. We first demonstrate how we measure the cosine similarity of two neurons. Then, we argue why this choice has been made and how it relates to the Hebbian Learning theory. We measure the similarity of two neurons p and q as:

$$Sim_{p,q}^l = \left| \frac{\boldsymbol{A}_{:,p}^{l-1} \cdot \boldsymbol{A}_{:,q}^l}{\left\|\boldsymbol{A}_{:,p}^{l-1}\right\| \left\|\boldsymbol{A}_{:,q}^l\right\|} \right|, \tag{2.2}$$

where $\boldsymbol{Sim}^l$ is the similarity matrix between neurons in two successive layers $l-1$ and l. $\boldsymbol{A}_{:,p}^{l-1}$ and $\boldsymbol{A}_{:,q}^l \in \mathbb{R}^m$ are the activation vectors corresponding to neurons p and q in layers $l-1$ and l, respectively. If $Sim_{p,q}^l$ is high for two unconnected neurons (close to 1), it means that they have a high similarity among their activations. Therefore, we prefer to add a connection between them as it suggests that this path contains important information about data. However, if $Sim_{p,q}^l$ is low for two neurons (close to 0), it means that the activations of neurons p and q are not similar, and the connection among them might not be beneficial for the network.

We now argue why cosine similarity can be used to measure the importance of a non-existing connection in sparse neural networks and how it connects to the Hebbian learning theory. Basically, by taking inspiration from the Hebbian

learning theory, we aim to rewire the neurons that fire together in the context of sparse training algorithms, instead of only strengthening the existing connections among neurons that fire together [Sch21]. It has been discussed in [Sch21] that connecting a pair of neurons with strong coincident activations can be viewed as a natural extension of Hebbian learning. Essentially, it is necessary to wire the neurons that usually fire together in order to understand better the relationship among the higher-order representation of those neurons. If a causal connection between their higher-order representation does exist, growing a connection among them will enable an effective inference about the relationship between them. Therefore, we need to discover which pairs of neurons usually fire together and then rewire them.

We employ cosine similarity to measure the relation between the activation values of two neurons. Such as Hebb's rule (Section 2.2.2), the importance of a connection in our method is also determined by multiplying the activations of its corresponding neurons, albeit normalized. In Equation 2.2, $\boldsymbol{A}^{l-1}_{:,p}$ is the pre-synaptic activation and $\boldsymbol{A}^{l}_{:,q}$ is the post-synaptic activation. If the activations of two connected neurons agree, by computing the dot product of activations, both Equation 2.2 and Hebb's rule assign higher importance to the corresponding connection. This would result in increased weight and a better chance of adding this connection. Thus, both methods reward connections between neurons that exhibit similar behavior. As mentioned earlier, the main difference between Hebb's rule and Equation 2.2 is normalization. We will discuss in Section 2.5.3 why the normalization step is necessary for evolving the topology of a sparse neural network.

In summary, if the cosine similarity of the activation vector of two neurons is high, it indicates the necessity of the connection between them in the network's performance. Therefore, we use the cosine similarity information to find out if the link between a pair of neurons should be rewired or not. Based on this knowledge, we propose two new algorithms to evolve the sparse neural network in the following sections.

2.3.3 Sequential Cosine Similarity-based and Random Topology Exploration (CTRE_{seq})

Our first proposed algorithm, Sequential Cosine Similarity-based and Random Topology Exploration (CTRE_{seq}) evolves the network topology using both cosine similarity between neurons of each pair of consecutive layers in the network and random search. Overall, in the beginning, at each training epoch, it removes

Algorithm 1 CTRE_{seq}

Input: Dataset $\mathbb{X}$, sparsity hyperparameter ε, drop fraction ζ, early stop epoch $e_{earlystop}$
Initialize the network with sparsity determined by ε, $flag_{randomsearch} = False$
for $i \in \{1,\ldots,\#epochs\}$ **do**
    Perform standard feed-forward and back-propagation
    for $l \in \{1,\ldots,H\}$ **do**
        Remove ζN^l of the weights with the smallest magnitude.
        if $flag_{randomsearch}$ **then**
            Add ζN^l connections randomly
        else
            Compute Similarity matrix $\boldsymbol{Sim}^l$ according to Equation 2.2
            Add ζN^l connections with the highest similarity value in $\boldsymbol{Sim}^l$
    if Accuracy on validation set does not improve in $e_{earlystop}$ **then**
        Set $flag_{randomsearch} = True$

unimportant connections based on their magnitude and adds new connections to the network based on their cosine similarity. When the network performance stops improving, the algorithm switches to a random topology search. In the following, we will explain the algorithm in more detail.

After initializing the sparse network with a sparsity level determined by ε, the training begins. The training procedure consists of two consecutive phases:

1. *Cosine Similarity-based Exploration*: The training starts with this phase in which each epoch includes three steps:

    (a) Firstly, a standard feed-forward and back-propagation are performed.

    (b) Then, a proportion ζ of connections with the lowest magnitude in each layer is removed. In Section 2.5.2, we further discuss why this choice has been made.

    (c) Subsequently, we add new connections to the network based on the neurons' similarity. Taking advantage of the cosine similarity metric, we measure the similarity of two neurons as formulated in Equation 2.2. In each layer, we add connections (as many connections as the removed connections in this layer) with the highest similarity between the corresponding neurons. The new connections are initialized with a small value from a uniform distribution.

2. *Random Exploration*: The second phase begins when the performance of the network on a validation set does not improve in $e_{earlystop}$ epochs ($e_{earlystop}$ is a hyperparameter of CTRE$_{\text{seq}}$). This is due to the fact that the activation values might not change significantly after some epochs and, consequently, the similarity of neurons. As a result, the topology search using cosine similarity might stop as well. To prevent this, we begin a random search when the classification accuracy on the validation set stops increasing. This phase is almost similar to phase 1, but they are different in the weight-regrowing policy. In this phase, instead of using cosine similarity information, we add connections randomly to the network. In this way, we prevent the early stopping of the topology search. Algorithm 1 summarizes this method.

2.3.4 Simultaneous Cosine Similarity-based and Random Topology Exploration (CTRE$_{\text{sim}}$)

To constantly exploit the cosine similarity information during training and avoid the early stopping of topology exploration, we propose another method for obtaining a sparse neural network, named Simultaneous Cosine Similarity-based and Random Topology Exploration (CTRE$_{\text{sim}}$).

Prior to the training, we initialize a sparse neural network. After that, the training procedure starts with three steps in each epoch. The first two steps are similar to CTRE$_{\text{seq}}$.

(a) Standard feed-forward and back-propagation

(b) Magnitude-based weight removal

(c) In this step, instead of relying solely on cosine similarity information or random addition, we combine both strategies. There are two reasons behind this choice: (1) As discussed in Section 2.3.3, as the training proceeds, the activation values become stable and might not change significantly after a while and, consequently, the similarity values. In CTRE$_{\text{seq}}$, we addressed this issue by switching completely to random search. However, the training speed might slow down if we rely only on the random search. (2) If we rely only on cosine similarity information, there is a possibility to add some connections based on the similarity of the neurons, which have been removed based on the magnitude in the weight removal step. It means that in these cases, the path between these pairs of similar neurons does not

Algorithm 2 CTRE_{sim}

Input: Dataset $\mathbb{X}$, sparsity hyperparameter ε, drop fraction ζ
Initialize the network with sparsity determined by ε
for $i \in \{1, \ldots, \#epochs\}$ **do**
    perform standard feed-forward and back-propagation
    for $l \in \{1, \ldots, H\}$ **do**
        Remove ζN^l of the weights with smallest magnitude.
        Compute Similarity matrix $\boldsymbol{Sim}^l$ according to Equation 2.2
        C_{sim} = Set of ζN^l connections with the highest similarity value in $\boldsymbol{Sim}^l$
        for each $c \in C_{sim}$ **do**
            if c was removed in the last weight removal step **then**
                Add a random connection to the network
            else
                Add connection c to the network

contribute to the performance of the network. Therefore, we should not add such connections to the network. These are the potential limitations of CTRE_{seq}.

To address these limitations, CTRE_{sim} takes another approach to prevent adding the removed connections which have a high cosine similarity to the network, as follows. In step *c*, we add the connections with high similarities to the network. However, if some connections with high cosine similarity are earlier removed based on their magnitude in step *b*, we add random connections to the network. In other words, we split our budget between similarity-based and random exploration. More importantly, we let the network dynamically decide how much budget should be allocated to each exploration at each epoch. The benefits from this approach are twofold; we prevent early stopping of the topology search and also prevent re-adding connections that have shown to be unhelpful for the network's performance. Algorithm 2 summarizes this method.

2.4 Experiments and Results

In this section, we evaluate our proposed algorithms and compare them with several state-of-the-art algorithms for obtaining a sparse neural network. First,

we describe the settings of the conducted experiments, including the hyperparameter values, implementation details, and datasets. Then, we compare them in terms of classification accuracy on several datasets and networks with different sizes and sparsity levels.

2.4.1 Settings

This section gives a brief overview of the experiment settings, including hyperparameter values, implementation details, and datasets used for the evaluation of the methods.

Hyperparameters

The network that we use to perform experiments is a 3-layer MLP as described in Section 2.3.1. The activation functions used for hidden and output layers are *Relu* and *Softmax*, respectively, and the loss function used is *CrossEntropy*. All the experiments are performed using 500 training epochs. The values for most hyperparameters have been selected using a grid search over a limited number of values. Each hyperparameter was tuned independently and less than 10 values was tested for each of the hyperparameters. The hyperparameter ζ has been set to 0.2. In Algorithm 1, $e_{early\,stop}$ has been set to 40. We train the network with Stochastic Gradient Decent (SGD) with momentum and L_2 regularizer. The momentum coefficient, the regularization coefficient, and the learning rate are 0.9, 0.0001, and 0.01, respectively. The datasets have been preprocessed using the Min-Max Scaler so that each feature is normalized between 0 and 1, except for Madelon, where we use a standard scaler (each feature will have zero mean and unit variance). For the image datasets, data augmentation has not been performed unless it has been explicitly stated.

Comparison

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

- **SNIP** [LAT19]. Single-shot network pruning (SNIP) is a dense-to-sparse sparsification algorithm that prunes the network prior to initialization based on connection sensitivity. It calculates this metric after a few iterations of dense training. After pruning, SNIP starts the training with the sparse neural network.

- **RigL** [EGM+20]. The rigged lottery (RigL) is a sparse-to-sparse algorithm for obtaining a sparse neural network that uses gradient information as the weight addition criteria.

- **SET** [MMS+18]. Sparse evolutionary training (SET) is a sparse-to-sparse training algorithm that uses random weight addition for updating the topology.

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

SET and RigL are recognized as seminal works in the dynamic sparse training field, each employing distinct growth criteria. They serve as foundational benchmarks for comparative analyses, as evidenced by numerous prior studies [YMN+21, LCC+21, LYMP21]. This choice is primarily driven by our specific aim to evaluate various weight growth strategies. In this chapter of the doctoral thesis, our objective is to compare the cosine-based weight growth strategy (CTRE) with the purely random (SET) and gradient-driven (RigL) approaches to weight growth. Thus, the main point of differentiation among these methods lies in their respective weight growth strategies. While other methodologies may incorporate additional components, we refrain from their examination as it falls outside the scope of our study. This focused approach ensures a clear evaluation of weight growth strategies within the context of our research.

SNIP stands as a seminal work in pruning prior to training, commonly employed as a benchmark in dynamic sparse training (DST) research. Its utilization of training solely with sparse matrices renders it a pertinent baseline in our studies.

Implementation

We evaluate our proposed methods and the considered baselines on eight datasets. We implemented our proposed method using Tensorflow [AAB+15]. The baseline of this implementation is the RigL code from Github [2]. It also includes the implementation of SNIP, SET, and fully-connected MLP. This code uses a binary mask over weights to implement sparsity. In addition, we provide a purely sparse implementation that uses SciPy library sparse matrices. This code is developed from the sparse implementation of SET, which is available on

[2]The implementation of RigL, SNIP, and SET is available at *https://github.com/google-research/rigl*.

Table 2.1: Datasets characteristics.

Dataset	Dimensions	Type	Samples	Train	Test	Classes
Isolet	617	Speech	7737	6237	1560	26
Madelon	500	Artificial	2600	2000	600	2
MNIST	784	Image	70000	60000	10000	10
Fashion_MNIST	784	Image	70000	60000	10000	10
CIFAR10	3072	Image	60000	50000	10000	10
CIFAR100	3072	Image	60000	50000	10000	100
PCMAC	3289	Text	1943	1554	389	2
BASEHOCK	4862	Text	1993	1594	399	2

Github[3]. For all the experiments, we use the Tensorflow implementation to have a fair comparison among methods. However, we provide the results using the sparse implementation in Appendix A.2. Most experiments were run on a *Dell R730* CPU. For image datasets, we used a *Tesla-P100* GPU. All the experiments were repeated with three random seeds. The only exception is the experiments from Section 2.4.2 where we run 15 random seeds to analyze the statistical significance of the obtained results with respect to the considered algorithms (Section 2.4.2). To ensure a fair comparison, for the sparse training methods (SET, RigL, and CTRE), the sparsity mask is updated at the end of each epoch, and the drop fraction (ζ) and learning rate are constant during training.

Datasets

We conducted our experiments on eight benchmark datasets as follows:

- **Madelon** [GGNZ08] is an artificial dataset with 20 informative features and 480 noise features.
- **Isolet** [FC91] has been created with the spoken name of each letter of the English alphabet.
- **MNIST** [LeC98] is a database of 28×28 images of handwritten digits.
- **Fashion-MNIST** [XRV17] is a database of 28×28 images of Zalando's articles.

[3]The pure sparse implementation of SET can be found on *https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks*.

- **CIFAR-10/100** [KH+09] are two datasets of 32×32 colour images categorized in 10/100 classes.
- **PCMAC & BASEHOCK** [Lan95] are two subsets of the 20 Newsgroups data.

The selection of these datasets has been meticulously curated to encompass a broad spectrum of data types. Specifically, we have chosen datasets representing four distinct categories: speech, image, text, and artificial data. Given our focus on MLP networks, it's imperative not only to demonstrate classification performance on image datasets but also to evaluate the model across other datasets common for MLP networks such as text, tabular, and speech data. This deliberate inclusion allows us to assess the generalizability and applicability of our proposed approach across various domains of application. More details about the datasets are presented in Table 2.1.

2.4.2 Performance Evaluation

In this experiment, we compare the methods in terms of classification accuracy on networks with varying sizes and sparsity levels. We consider three MLPs, each having three hidden layers with 100, 500, and 1000 hidden neurons, respectively. By changing the value of ε for each MLP, we study the effect of sparsity level on the performance of the methods. Table 2.2 summarizes the results of these experiments that are carried out on the five datasets, including tabular and image datasets that have different characteristics. We have also included the density (in percentage) and the number of connections (divided by 10^3) for each network in this table. For training on each dataset, we allocate 10% of the training set to a validation set. During training, each MLP is trained on the new training set. At each epoch, we measure the performance on the validation set. Finally, Table 2.2 presents the results of each algorithm on an unseen test set and uses the model that gives the highest validation accuracy during training. The learning curves regarding each case are presented in Appendix A.1; however, we present some interesting cases in Figure 2.2.

First, we analyze the performance of methods on the two tabular datasets. As can be seen in Table 2.2, on the Madelon dataset, CTRE_{sim} is the best performer in most cases. Interestingly, the accuracy increases when the network becomes sparser. However, this can be explained intuitively; since the Madelon dataset contains many noise features ($> 95\%$), the higher the number of the connections is, the higher the risk for over-fitting the noise features will be. CTRE_{sim} can find

the most important information paths in the network, which most likely start from the input neurons corresponding to the informative features. As a result, it can reach an accuracy of 78.8% with only 0.3% of total connections of the equivalent dense network ($n^l = 1000$), while the maximum accuracy achieved by other methods considered is 61.9% (SET). On the second tabular dataset, Isolet, CTRE_{sim} is the best performer on two very sparse models, including 0.4% ($n^l = 500$) and 0.3% ($n^l = 1000$) densities. In addition, in all the other cases, CTRE_{sim} and CTRE_{seq} are the second and third-best performers. In terms of learning speed, we can observe in Figure 2.2 that CTRE_{sim} can find a good topology much faster than other methods, which results in an increase in the accuracy within a short period after the training starts. From Figure 2.2, it can be seen that RigL fails to find an informative sub-network in these cases ($D < 0.3\%$). This indicates that gradient information might not be informative in highly sparse networks.

On the image dataset, CTRE_{sim} and CTRE_{seq} are the best and second-best performers in most of the cases considered. When the network size is small ($n^l = 100$), SET is the major competitor of CTRE. However, when the model size increases, CTRE outperforms SET. This indicates that the pure random weight addition policy in SET can perform well in networks with a higher density, while it is hard to find such sub-network randomly in high-sparsity scenarios due to the

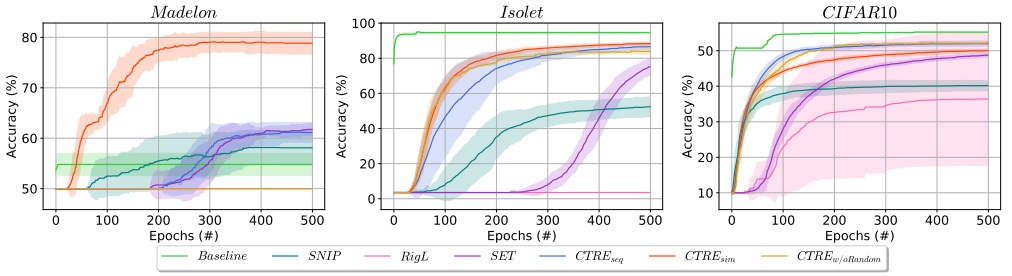


Figure 2.2: Classification accuracy (%) comparison among methods on a highly large and sparse 3-layer MLP with a density lower than 0.3% ($n^l = 1000$, $\varepsilon = 1$). The shaded areas show the standard deviation of the results. CTRE shows high learning speed among various datasets. On the Madelon dataset, CTRE_{sim} is able to achieve a decent performance which is due to combining cosine-based search and random search.

Table 2.2: Classification accuracy (%) comparison among methods on networks with various sizes and sparsity levels. The density (%) and number of connections for each case is indicated in the table. Please note that N (total number of parameters of the network) is scaled by $\times 10^3$.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ϵ			ϵ			ϵ		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	Baseline ($N(\times 10^3)$)	54.9 ± 1.0 (70.2)			54.9 ± 1.0 (751.0)			54.9 ± 1.0 (2502.0)		
	D(%) ($N(\times 10^3)$)	1.6 (1.1)	7.8 (5.5)	20.4 (14.3)	0.5 (3.5)	2.3 (17.5)	6.1 (45.5)	0.3 (6.5)	1.3 (32.5)	3.4 (84.5)
	SNIP	56.5 ± 3.5	56.1 ± 2.9	54.1 ± 2.6	57.3 ± 3.0	56.4 ± 2.5	57.5 ± 1.9	58.1 ± 3.4	58.4 ± 1.5	57.8 ± 1.6
	RigL	60.4 ± 3.1	59.7 ± 1.8	59.3 ± 2.2	51.3 ± 3.5	61.9 ± 2.0	60.0 ± 2.0	50.0 ± 0.0	61.5 ± 3.0	61.1 ± 1.7
	SET	60.4 ± 2.1	57.8 ± 3.2	58.1 ± 1.7	61.4 ± 2.6	59.4 ± 2.4	57.9 ± 3.9	61.7 ± 1.2	59.0 ± 1.8	58.2 ± 2.3
	$CTRE_{seq}$	**82.2 ± 2.4**	72.5 ± 2.0	63.9 ± 1.2	61.2 ± 2.4	**81.5 ± 1.4**	71.8 ± 2.0	61.1 ± 1.9	**83.9 ± 2.0**	**76.5 ± 1.9**
	$CTRE_{sim}$	81.6 ± 1.3	**73.0 ± 1.6**	**65.6 ± 3.0**	**79.4 ± 1.7**	77.7 ± 1.4	**73.0 ± 1.5**	**78.8 ± 2.2**	78.5 ± 1.0	74.6 ± 1.4
Isolet	Baseline ($N(\times 10^3)$)	94.4 ± 0.1 (84.3)			94.3 ± 0.0 (821.5)			94.6 ± 0.4 (2643.0)		
	D(%) ($N(\times 10^3)$)	1.5 (1.2)	7.4 (6.2)	19.2 (16.2)	0.4 (3.6)	2.2 (18.2)	5.8 (47.4)	0.3 (6.6)	1.3 (33.2)	3.3 (86.4)
	SNIP	48.9 ± 5.8	90.1 ± 1.1	92.8 ± 0.7	49.2 ± 8.6	89.9 ± 1.0	92.3 ± 0.6	52.4 ± 5.6	89.7 ± 0.8	92.0 ± 0.6
	RigL	66.0 ± 28.6	90.0 ± 0.8	92.3 ± 0.7	3.5 ± 0.0	88.4 ± 1.0	91.9 ± 1.2	3.5 ± 0.0	87.2 ± 1.8	91.2 ± 1.5
	SET	**89.1 ± 1.2**	**94.2 ± 0.7**	**94.9 ± 0.4**	86.5 ± 1.2	**93.2 ± 0.7**	**94.7 ± 0.5**	75.2 ± 4.7	**92.9 ± 0.5**	**94.4 ± 0.8**
	$CTRE_{seq}$	86.7 ± 1.8	92.4 ± 1.1	94.3 ± 0.5	87.2 ± 2.0	92.3 ± 0.6	94.0 ± 0.4	86.7 ± 1.6	91.5 ± 1.0	93.7 ± 0.5
	$CTRE_{sim}$	87.5 ± 0.8	93.4 ± 0.7	94.3 ± 0.8	**87.8 ± 1.1**	91.7 ± 1.1	93.9 ± 0.5	**88.3 ± 0.7**	91.3 ± 1.3	93.1 ± 0.6
MNIST	Baseline ($N(\times 10^3)$)	97.9 ± 0.0 (99.4)			98.2 ± 0.1 (897.0)			98.2 ± 0.1 (2794.0)		
	D(%) ($N(\times 10^3)$)	1.4 (1.4)	7.0 (7.0)	18.2 (18.1)	0.4 (3.8)	2.1 (19.0)	5.5 (49.3)	0.2 (6.8)	1.2 (34.0)	3.2 (88.3)
	SNIP	91.1 ± 0.5	96.3 ± 0.2	97.2 ± 0.1	93.8 ± 0.3	96.6 ± 0.2	97.2 ± 0.2	94.4 ± 0.3	96.7 ± 0.2	97.3 ± 0.2
	RigL	94.5 ± 0.3	96.4 ± 0.2	97.2 ± 0.2	95.9 ± 0.3	96.7 ± 0.2	97.0 ± 0.1	96.0 ± 0.2	96.6 ± 0.2	96.9 ± 0.2
	SET	95.6 ± 0.3	97.1 ± 0.1	97.6 ± 0.1	95.9 ± 0.2	97.4 ± 0.1	97.8 ± 0.1	95.8 ± 0.1	97.4 ± 0.2	97.7 ± 0.1
	$CTRE_{seq}$	**95.7 ± 0.2**	**97.3 ± 0.2**	97.7 ± 0.1	**97.0 ± 0.2**	97.6 ± 0.2	97.8 ± 0.1	**97.3 ± 0.1**	**97.7 ± 0.1**	97.8 ± 0.1
	$CTRE_{sim}$	95.5 ± 0.2	**97.3 ± 0.1**	**97.8 ± 0.1**	96.4 ± 0.2	**97.7 ± 0.1**	**98.0 ± 0.1**	96.6 ± 0.2	**97.7 ± 0.1**	**97.9 ± 0.1**
Fashion-MNIST	Baseline ($N(\times 10^3)$)	88.3 ± 0.2 (99.4)			89.8 ± 0.1 (897.0)			90.1 ± 0.1 (2794.0)		
	D(%) ($N(\times 10^3)$)	1.4 (1.4)	7.0 (7.0)	18.2 (18.1)	0.4 (3.8)	2.1 (19.0)	5.5 (49.3)	0.2 (6.8)	1.2 (34.0)	3.2 (88.3)
	SNIP	79.5 ± 2.3	86.0 ± 0.3	87.1 ± 0.3	81.1 ± 1.2	86.3 ± 0.3	87.4 ± 0.4	82.2 ± 1.1	86.5 ± 0.2	87.3 ± 0.2
	RigL	84.8 ± 0.2	86.4 ± 0.3	87.2 ± 0.2	86.1 ± 0.2	86.4 ± 0.3	86.5 ± 0.3	86.1 ± 0.3	86.3 ± 0.4	86.6 ± 0.4
	SET	**85.8 ± 0.3**	87.4 ± 0.4	87.8 ± 0.3	86.5 ± 0.2	87.6 ± 0.2	88.0 ± 0.2	86.3 ± 0.2	87.8 ± 0.3	87.8 ± 0.2
	$CTRE_{seq}$	**85.8 ± 0.5**	**87.5 ± 0.3**	87.4 ± 0.3	**87.1 ± 0.4**	87.9 ± 0.3	88.0 ± 0.2	**87.3 ± 0.2**	88.0 ± 0.3	**88.3 ± 0.2**
	$CTRE_{sim}$	**85.8 ± 0.3**	**87.5 ± 0.3**	**88.0 ± 0.2**	86.4 ± 0.5	**88.1 ± 0.2**	**88.3 ± 0.2**	86.5 ± 0.3	**88.1 ± 0.3**	**88.3 ± 0.2**
CIFAR10	Baseline ($N(\times 10^3)$)	51.2 ± 0.5 (328.2)			53.2 ± 0.2 (2041.0)			55.2 ± 0.2 (5082.0)		
	D(%) ($N(\times 10^3)$)	1.1 (3.7)	5.6 (18.4)	14.6 (47.9)	0.3 (6.1)	1.5 (30.4)	3.9 (79.1)	0.2 (9.1)	0.9 (45.4)	2.3 (118.1)
	SNIP	35.8 ± 3.8	47.6 ± 0.7	49.5 ± 0.7	39.4 ± 1.8	48.9 ± 0.6	50.7 ± 0.6	40.2 ± 1.4	49.3 ± 1.0	50.9 ± 0.6
	RigL	46.0 ± 0.6	49.3 ± 0.6	50.5 ± 0.5	46.4 ± 9.8	50.0 ± 1.0	49.9 ± 1.1	36.4 ± 18.7	50.3 ± 1.1	49.9 ± 0.8
	SET	**48.3 ± 0.5**	49.9 ± 0.5	**50.6 ± 0.5**	49.4 ± 0.3	50.6 ± 0.4	51.4 ± 0.6	48.8 ± 0.5	50.6 ± 0.4	51.1 ± 0.6
	$CTRE_{seq}$	47.9 ± 0.8	50.1 ± 0.5	50.1 ± 0.5	**51.3 ± 0.5**	**52.7 ± 0.5**	**52.6 ± 0.6**	**52.0 ± 0.7**	**53.2 ± 0.6**	**53.6 ± 0.6**
	$CTRE_{sim}$	48.2 ± 0.4	**50.3 ± 0.3**	50.5 ± 0.4	50.6 ± 0.4	52.6 ± 0.7	52.5 ± 0.7	50.0 ± 0.4	52.7 ± 0.5	53.5 ± 0.5
CIFAR100	Baseline ($N(\times 10^3)$)	23.1 ± 0.6 (337.2)			23.1 ± 0.6 (2086.0)			23.1 ± 0.6 (5172.0)		
	D(%) ($N(\times 10^3)$)	1.1 (3.8)	5.6 (18.9)	14.5 (49.0)	0.3 (6.2)	1.5 (30.9)	3.8 (80.2)	0.2 (9.2)	0.9 (45.9)	2.3 (119.2)
	SNIP	5.9 ± 0.6	15.7 ± 0.7	20.4 ± 0.3	6.6 ± 0.8	17.9 ± 0.7	22.1 ± 0.4	6.2 ± 0.8	18.3 ± 0.6	22.5 ± 0.5
	RigL	7.4 ± 4.4	19.9 ± 0.5	21.4 ± 0.4	1.0 ± 0.0	21.0 ± 0.5	21.4 ± 0.5	1.0 ± 0.0	1.0 ± 0.0	20.4 ± 0.6
	SET	**14.7 ± 0.4**	20.3 ± 0.3	21.7 ± 0.3	14.3 ± 1.5	22.7 ± 0.3	24.1 ± 0.4	1.0 ± 0.0	23.3 ± 0.3	24.5 ± 0.3
	$CTRE_{seq}$	12.7 ± 0.7	**21.1 ± 0.3**	21.8 ± 0.5	**18.7 ± 0.4**	**23.6 ± 0.4**	24.0 ± 0.4	**21.4 ± 0.4**	**23.9 ± 0.5**	24.7 ± 0.4
	$CTRE_{sim}$	13.8 ± 0.4	20.6 ± 0.4	**21.9 ± 0.4**	17.0 ± 0.3	23.0 ± 0.3	**24.6 ± 0.4**	17.3 ± 0.4	23.5 ± 0.3	**25.1 ± 0.3**

very large search space. RigL also has comparable performance to SET, except for very sparse models. As discussed in the previous paragraph, on a highly sparse network ($D < 0.3\%$), RigL has poor performance. Besides, as shown in Figure 2.2, SNIP starts with a steep increase in accuracy due to the few iterations of training a dense network and thus, starting with good topology. However, as the training proceeds, this topology cannot achieve the same performance as other methods. Therefore, it indicates that dynamic weight update is an essential factor in the sparse training of neural networks.

These observations confirm that cosine similarity is an informative criterion for adding weight to the network compared to random (SET) and gradient-based addition (RigL) in very sparse neural networks. CTRE can reach a better performance than state-of-the-art sparse training algorithms in terms of learning speed and accuracy when the network is highly sparse. Besides, by comparing the results with the dense network, it is clear that it is possible to reach a comparable performance to the dense network even with a network with 100 times fewer connections which is an excellent choice for low-resource devices on edge. We further compare the learning speed of the algorithms in Appendix A.1.2 and their computational complexity in Appendix A.1.3.

Statistical Significance Analysis

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

We perform the KS-test between the results obtained by CTRE (for simplicity, we consider maximum results of $CTRE_{seq}$ and $CTRE_{sim}$) and the other considered algorithms for the experiments in Table 2.2. The results of the KS-test are summarized in Table 2.3. In this table, *Reject* shows that the results are sufficiently distinct, and *True* means that the obtained results are close together. The * sign in Table 2.3 shows that an algorithm has achieved the maximum accuracy in the corresponding experiment. Finally, the entries colored red show an experiment where a compared method obtains a close result to CTRE while having lower mean accuracy.

From Table 2.3, we can observe that in the majority of the experiments, CTRE obtains higher mean accuracy than the other methods while being statistically

Table 2.3: Statistical significance of the results. Each entry shows the result of the KS-test among the results of CTRE and the compared methods for a specific network size and sparsity level. *Reject* shows that the results are distinct, and *True* indicates that the results are close together. The * sign shows that an algorithm has achieved the maximum accuracy in the corresponding experiment.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
			ϵ			ϵ			ϵ	
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	SNIP	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	RigL	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	SET	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	CTRE	-*	-*	-*	-*	-*	-*	-*	-*	-*
Isolet	SNIP	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	RigL	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	SET	Reject*	Reject*	Reject*	Reject	Reject*	Reject*	Reject	Reject*	Reject*
	CTRE	-	-	-	-*	-	-	-*	-	-
MNIST	SNIP	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	RigL	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	SET	True	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	CTRE	-*	-*	-*	-*	-*	-*	-*	-*	-*
Fashion-MNIST	SNIP	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	RigL	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	SET	True*	True	True	Reject	Reject	Reject	Reject	True	Reject
	CTRE	-*	-*	-*	-*	-*	-*	-*	-*	-*
CIFAR10	SNIP	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	RigL	Reject	Reject	True	Reject	Reject	Reject	Reject	Reject	Reject
	SET	True*	Reject	True*	Reject	Reject	Reject	Reject	Reject	Reject
	CTRE	-	-*	-	-*	-*	-*	-*	-*	-*
CIFAR100	SNIP	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	RigL	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
	SET	Reject*	Reject	True	Reject	Reject	Reject	Reject	Reject	Reject
	CTRE	-	-*	-*	-*	-*	-*	-*	-*	-*

different from them. The only dataset where the results in most cases are close is the Fashion-MNIST dataset where SET has comparable results to CTRE. In addition, in high sparsity regime and large network size ($n^l = 1000$, $\varepsilon = 1$), CTRE achieves the highest accuracy among the methods while being significantly distinct from them. Overall, Table 2.3 indicates that CTRE is a well-performing algorithm in terms of classification accuracy that achieves significantly different results from the other methods.

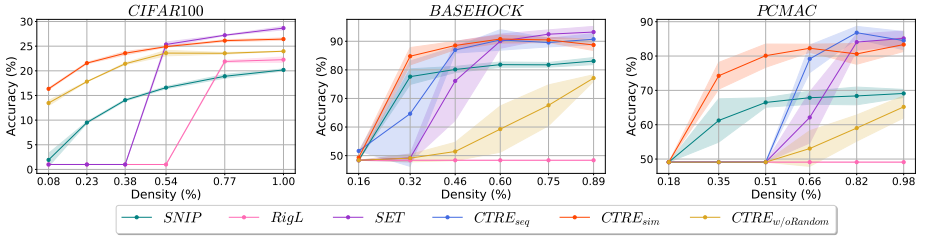


Figure 2.3: Sparsity-accuracy trade-off on highly sparse neural networks on three datasets.

2.4.3 Sparsity-Performance Trade-off Analysis in Highly Sparse MLPs

We carry out another experiment to study the trade-off between sparsity and accuracy on very high sparsity cases. We perform this experiment for two difficult classification tasks including, image classification on CIFAR-100, which is considered a more difficult dataset than the earlier considered image datasets, and text classification on PCMAC and BASESHOCK which are subsets of the 20-newsgroup dataset; they have a high number of features and a low number of samples. This experiment uses a 3-layer MLP with 1000 and 3000 hidden neurons for text datasets and the CIFAR-100 dataset, respectively. We change the density value between 0 and 1 and compare our proposed approaches to SNIP, RigL, and SET (due to the close performance of CTRE_{sim} and CTRE_{seq} on earlier considered image datasets, on CIFAR-100, we perform the experiments with CTRE_{sim}). We use data augmentation for CIFAR-100. Also, as the network is considerably large on this dataset, we set the learning rate to 0.05 to speed up the training. The results are presented in Figure 2.3.

As shown in Figure 2.3, in highly sparse networks ($D < 0.5\%$), CTRE_{sim} outperforms other methods by a large gap. As discussed in Section 2.4.2, RigL performs poorly in these scenarios. SNIP outperforms SET and RigL at very low densities while still having lower results than CTRE_{sim} in all cases. While SET outperforms other methods for larger density values on CIFAR-100 and BASESHOCK, it performs poorly on a very sparse network. On text datasets, CTRE_{seq} has comparable performance to CTRE_{sim} and SET on higher densities, and it achieves the highest accuracy on PCMAC. Overall, we can observe that CTRE_{sim} has decent performance on these three datasets with a density value

between 0.3% and 0.5%.

2.5 Discussion

In this section, we perform an in-depth analysis to understand the behavior of CTRE better. First, in Section 2.5.1, we perform two ablation studies to study the effectiveness of both random topology search and similarity importance metric in the performance of CTRE. In Section 2.5.2, we discuss why we have chosen magnitude over cosine similarity for the weight removal step. In Section 2.5.3, we discuss why the insensitivity of cosine similarity to the vector's magnitude is important in the performance of CTRE. Finally, we discuss the convergence of CTRE in Section 2.5.4.

2.5.1 Ablation Study: Analysis of Topology Search Policies

This section presents and discusses the results of two ablation studies designed to understand better the effect of different topology search policies in CTRE. In the following, we describe each ablation experiment separately.

Ablation Study 1: Random Topology Search

The first ablation study aims to analyze the effect of random connection addition on the behavior of CTRE. Therefore, instead of using the similarity information and random search (simultaneously in CTRE_{sim} and sequentially in CTRE_{seq}), we only use the cosine similarity information at each epoch. We call this approach $\text{CTRE}_{w/oRandom}$ and repeat the experiments from Section 2.4.2. The detailed results are available in Table 2.4.

As can be seen in Table 2.4, in most cases considered, $\text{CTRE}_{w/oRandom}$ has been outperformed by CTRE_{sim} and CTRE_{seq}. On the other hand, we can observe that on image datasets, $\text{CTRE}_{w/oRandom}$ has comparable performance to the other two methods. This indicates the effectiveness of similarity information on the image datasets. However, on tabular datasets, it performs poorly on high-sparsity cases ($\varepsilon = 1$). Therefore, using only cosine information in these scenarios can cause the topology search to be stuck in a local minimum. This might have originated from the early stopping of changes in the activation values, which leads to an early stop in the topology search. CTRE_{seq} solves this by changing the weight update policy to random search. However, there is a risk of early switching to random search when the cosine information has not been

fully exploited. Finally, by considering both random and cosine information in each epoch, the CTRE_{sim} algorithm will minimize the risk of staying in the local minimum or switching to a completely random search, both of which might slow the training process. In the context of network topology search, these components can also be characterized as exploitation (local information based on the similarity between neurons) and exploration (random search). As a result, CTRE_{sim} can mitigate the limitations of CTRE_{seq} and find a performant sub-network by leveraging these two components, which outperform state-of-the-art algorithms.

Ablation Study 2: Cosine similarity-based Topology Search

To study the effectiveness of cosine similarity addition in the performance of CTRE, we design an experiment. In this experiment, we add connections in the reverse order of importance to the network. We expect that adding weights in this order would result in poor performance. We perform this experiment on CTRE_{sim}. Concretely, at each step, we add the weights with the lowest similarity

Table 2.4: Classification accuracy (%) comparison among Cosine similarity-based methods.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ϵ			ϵ			ϵ		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	CTRE_{seq}	**82.2±2.4**	72.5±2.0	63.9±1.2	61.2±2.4	**81.5±1.4**	71.8±2.0	61.1±1.9	**83.9±2.0**	**76.5±1.9**
	CTRE_{sim}	81.6±1.3	**73.0±1.6**	**65.6±3.0**	**79.4±1.7**	77.7±1.4	**73.0±1.5**	**78.8±2.2**	78.5±1.0	74.6±1.4
	$\text{CTRE}_{\text{w/oRandom}}$	79.6±2.0	59.0±2.3	57.6±1.2	58.6±12.0	72.7±5.3	61.4±1.6	50.0±0.0	71.2±1.4	56.3±3.7
	$\text{CTRE}_{\text{sim/LTH}}$	61.3±3.4	58.0±2.6	56.9±1.7	53.2±0.7	59.6±1.4	58.2±1.7	51.1±0.2	58.9±1.6	58.3±1.2
Isolet	CTRE_{seq}	86.7±1.8	92.4±1.1	94.3±0.5	87.2±2.0	**92.3±0.6**	94.0±0.4	86.7±1.6	91.5±1.0	93.7±0.5
	CTRE_{sim}	**87.5±0.8**	**93.4±0.7**	94.3±0.8	**87.8±1.1**	91.7±1.1	93.9±0.5	**88.3±0.7**	91.3±1.3	93.1±0.6
	$\text{CTRE}_{\text{w/oRandom}}$	66.7±11.3	91.9±0.3	93.5±0.2	84.5±0.8	89.9±0.3	92.8±0.5	83.9±1.9	87.6±1.1	92.1±0.9
	$\text{CTRE}_{\text{sim/LTH}}$	81.1±1.3	93.3±0.6	**95.0±0.2**	66.3±4.9	91.7±0.6	**94.6±0.4**	61.2±11.8	**91.8±1.1**	**93.8±0.2**
MNIST	CTRE_{seq}	**95.7±0.2**	**97.3±0.2**	97.7±0.1	**97.0±0.2**	97.6±0.2	97.8±0.1	**97.3±0.1**	**97.7±0.1**	97.8±0.1
	CTRE_{sim}	95.5±0.2	**97.3±0.1**	**97.8±0.1**	96.4±0.2	**97.7±0.1**	**98.0±0.1**	96.6±0.2	**97.7±0.1**	**97.9±0.1**
	$\text{CTRE}_{\text{w/oRandom}}$	94.8±0.3	97.0±0.2	97.5±0.1	96.9±0.0	97.4±0.1	97.4±0.1	97.2±0.3	97.5±0.2	97.5±0.1
	$\text{CTRE}_{\text{sim/LTH}}$	94.6±0.2	96.9±0.1	97.4±0.1	94.2±0.1	97.3±0.2	97.4±0.0	93.9±0.0	97.1±0.0	97.5±0.1
Fashion-MNIST	CTRE_{seq}	**85.8±0.5**	**87.5±0.3**	87.4±0.3	**87.1±0.4**	87.9±0.3	88.0±0.2	**87.3±0.2**	88.0±0.3	**88.3±0.2**
	CTRE_{sim}	**85.8±0.3**	**87.5±0.3**	**88.0±0.2**	86.4±0.5	**88.1±0.2**	**88.3±0.2**	86.5±0.3	**88.1±0.3**	**88.3±0.2**
	$\text{CTRE}_{\text{w/oRandom}}$	83.9±0.4	86.9±0.3	86.9±0.1	86.2±0.3	87.9±0.2	88.0±0.2	86.8±0.3	87.7±0.4	88.2±0.2
	$\text{CTRE}_{\text{sim/LTH}}$	85.0±0.2	87.3±0.3	87.6±0.2	84.4±0.3	87.5±0.3	87.8±0.3	84.2±0.2	87.6±0.2	88.0±0.2
CIFAR10	CTRE_{seq}	47.9±0.8	50.1±0.5	50.1±0.5	**51.3±0.5**	**52.7±0.5**	52.6±0.6	52.0±0.7	53.2±0.6	**53.6±0.6**
	CTRE_{sim}	**48.2±0.4**	**50.3±0.3**	**50.5±0.4**	50.6±0.4	52.6±0.7	52.5±0.7	50.0±0.4	52.7±0.5	53.5±0.5
	$\text{CTRE}_{\text{w/oRandom}}$	45.4±0.2	49.4±0.3	49.9±0.4	50.9±0.3	52.2±0.6	**52.7±0.4**	**52.3±0.2**	**53.3±0.5**	53.5±0.4
	$\text{CTRE}_{\text{sim/LTH}}$	46.7±0.5	49.6±0.5	50.4±0.3	45.4±0.5	50.9±0.2	51.2±0.4	44.9±0.5	50.6±0.1	50.9±0.3

among the corresponding neurons. If a weight with a very low similarity has been removed in the last weight removal step, we add a random connection instead. We call this method $\text{CTRE}_{\text{sim/LTH}}$ (LTH refers to low to high importance).

As can be seen in Table 2.4, $\text{CTRE}_{\text{sim/LTH}}$ has been outperformed by CTRE_{sim} and CTRE_{seq} in most of the cases considered. This shows that cosine similarity is a useful metric to detect the most important weights in the network. By comparing $\text{CTRE}_{\text{sim/LTH}}$ with SET (Table 2.2), it is clear that in most cases $\text{CTRE}_{\text{sim/LTH}}$ has a close or slightly worse accuracy than SET. Therefore, it can be inferred that $\text{CTRE}_{\text{sim/LTH}}$ is selecting non-informative weights, which can be similar to or worse than a random search. As a result, this can indicate the effectiveness of the introduced similarity metric (Equation 2.2) in finding a well-performing sparse neural network. It is worth noting that on the Isolet dataset, $\text{CTRE}_{\text{sim/LTH}}$ outperforms CTRE_{sim} and CTRE_{seq} in some cases, particularly in the networks with higher density. This is similar to the results of SET as well. Therefore, we can conclude that random search outperforms other methods on the Isolet dataset and low sparsity levels. However, it is not easy to find a highly sparse network using the random search policy.

2.5.2 Analysis of Weight Removal Policy

In this section, we aim to analyze the weight removal policy and further explain the reason behind choosing magnitude-based pruning over the cosine similarity (discussed in Section 2.3.2). In many previous studies [MMS+18, EGM+20], magnitude-based pruning has been commonly used as a criterion to remove unimportant weight from a neural network. We design an experiment to compare the performance of magnitude-based and cosine similarity-based pruning in neural networks.

In this experiment, we start with a trained network and gradually remove weights based on the magnitude and cosine similarity value (Using Equation 2.2) of the corresponding connection. We also consider random pruning as the baseline.

Settings. We perform this experiment using two networks: (1) A 3-layer dense MLP with 1000 neurons in each layer, and (2) A 3-layer sparse MLP with 1000 neurons in each layer that is trained using the SET approach [MMS+18] (3.2% density). The choice of SET instead of CTRE was made to avoid any biases on the cosine similarity weight removal, as CTRE uses cosine information to add weights. Both of these networks are trained on the MNIST dataset.

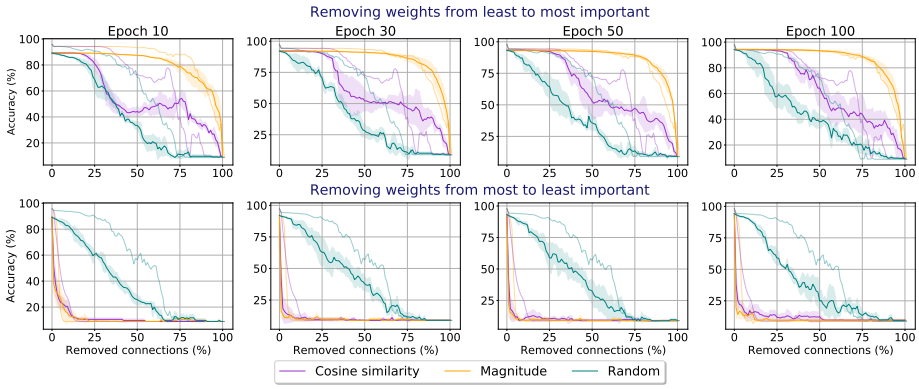


Figure 2.4: Effect of weight removal using three criteria including magnitude, cosine similarity, and random, on the classification accuracy (%) at different epochs. The lines with higher transparency correspond to the weight removal of the SET-MLP and the lines with lower transparency correspond to the dense-MLP

Weight Removal. We remove weights with two orders on each of the sparse and dense networks: least to most important and vice versa. We remove weights gradually such that at each step, we remove 1% of the connections and measure the accuracy of the pruned network until no connection remains in the network.

Results. The results when the two networks are trained for 10, 30, 50, and 100 epochs are available in Figure 2.4. In this figure, the lines with higher transparency correspond to the weight removal of the SET-MLP, and the lines with lower transparency correspond to the dense-MLP. This experiment has been repeated with three seeds for each case.

As shown in Figure 2.4, when weights are removed from least to most important, magnitude-based pruning can order weights better than cosine similarity-based pruning. When the networks are trained for 100 epochs, by dropping the unimportant weights using magnitude, the major accuracy drop starts almost after removing 70% of the connections, while it happens after removing 30% for cosine similarity. This behavior exists in both the dense and the sparse networks. As expected, the drop for random removal happens from the beginning of the pruning procedure. In earlier epochs (10, 30, and 50), the drop in the accuracy

happens earlier for both magnitude and cosine similarity.

It can be seen in Figure 2.4, by removing weights in the opposite order (from most to least important), the behavior of drop in the accuracy is almost similar for cosine similarity-based and magnitude-based pruning in SET-MLP, particularly in the earlier epochs. Therefore, both magnitude and cosine similarity can identify the most important connections in good order. However, this behavior is different in the dense network and magnitude-based pruning can better detect the most important weights. In the dense network, the drop in the accuracy for magnitude-based pruning happens earlier than cosine similarity pruning.

Conclusions. These observations can lead us to conclude that, firstly, the magnitude can be a good metric for weight removal in the dynamic sparse training framework. Secondly, it can be inferred that cosine similarity can be a good metric for adding the most important connections in the weight addition phase in sparse neural networks in the absence of magnitude. As discussed earlier, the cosine similarity information of each connection is an informative criterion to detect the most important weights in a sparse neural network and has similar behavior to magnitude-based pruning in these scenarios. Therefore, in the absence of magnitude for non-existing connections in a sparse neural network (during weight addition), cosine similarity can be a useful criterion to detect the most important weights without requiring computing dense gradient information.

2.5.3 Magnitude Insensitivity: The Favorable Feature of Cosine Similarity in Noisy Environments

This section further discusses why cosine similarity has been chosen as a metric to determine the importance of non-existing connections. Specifically, we focus on analyzing the importance of normalization in Equation 2.2 in the performance of the algorithm. While based on the Hebbian learning rule, the connection among a pair of neurons with high activations should be strengthened, we argue that in the search for a performant sparse neural network, the magnitude of the activations should be ignored.

Based on Hebb's rule (Section 2.2.2), the connection among the neurons with high activations receives higher synaptic updates. Therefore, if we evolve the topology using this rule (without any normalization) the importance of a non-existing connection should be determined by: $\left|\boldsymbol{A}^{l-1}_{:,p} \cdot \boldsymbol{A}^{l}_{:,q}\right|$. We evaluate the

Table 2.5: Classification accuracy (%) comparison of Cosine similarity-based methods and pure Hebbian-based evolution, on the Madelon dataset. Please note that N (total number of parameters of the network) is scaled by $\times 10^3$.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ϵ			ϵ			ϵ		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
	$D(\%)$ $(N(\times 10^3))$	1.6 (1.1)	7.8 (5.5)	20.4 (14.3)	0.5 (3.5)	2.3 (17.5)	6.1 (45.5)	0.3 (6.5)	1.3 (32.5)	3.4 (84.5)
Madelon	CTRE_{seq}	**82.2±2.4**	72.5±2.0	63.9±1.2	61.2±2.4	**81.5±1.4**	71.8±2.0	61.1±1.9	**83.9±2.0**	**76.5±1.9**
	CTRE_{sim}	81.6±1.3	**73.0±1.6**	**65.6±3.0**	**79.4±1.7**	77.7±1.4	**73.0±1.5**	**78.8±2.2**	78.5±1.0	74.6±1.4
	$\text{CTRE}_{\text{seq-Hebb}}$	63.1±1.0	58.8±1.8	60.1±0.9	60.8±3.3	59.6±0.3	59.4±0.8	61.6±2.9	56.8±2.1	58.0±0.6
	$\text{CTRE}_{\text{sim-Hebb}}$	59.6±3.1	56.9±7.9	60.1±3.0	58.8±4.2	62.7±1.0	59.1±1.1	60.2±4.3	59.1±2.1	59.4±3.3

performance of this metric by replacing it with Equation 2.2 in CTRE_{sim} and CTRE_{seq}; we name these algorithms $\text{CTRE}_{\text{sim-Hebb}}$ and $\text{CTRE}_{\text{seq-Hebb}}$, respectively.

We evaluate these methods on the Madelon dataset. The reason behind choosing this dataset is due to its interesting properties; it contains 480 noisy features (out of the 500 features). Therefore, finding informative information paths through the network is considered to be a challenging task. The settings of this experiment are similar to Section 2.4.2. We measure the performance on networks with different sizes and sparsity levels. The results are presented in Table 2.5 and the accuracy during training is plotted in Figure 2.5.

$\text{CTRE}_{\text{sim-Hebb}}$ and $\text{CTRE}_{\text{seq-Hebb}}$ have been outperformed by CTRE_{sim} and CTRE_{seq} in all cases considered. Particularly, we can observe that as the network becomes sparser, the gap between the performance of the pure Hebbian-based methods and the cosine similarity-based methods increases. The poor performance of $\text{CTRE}_{\text{sim-Hebb}}$ and $\text{CTRE}_{\text{seq-Hebb}}$ on the Madelon dataset resulted from their sensitivity to the magnitude of activation values. As Madelon contains many noisy features, some uninformative neurons likely receive a high activation value. Therefore, if we use only the activation magnitude to find the informative paths of information, the algorithm will be biased on the neurons with very high activation, which might not be informative. Therefore, it is likely to assign new connections to noisy features with high activation. This would cause the algorithm to be stuck in a local minimum which might be difficult to escape as these neurons continue to receive more and more connections at each epoch. Furthermore, as the networks become sparser, the informative features have a lower chance of receiving more connections (there are more noisy features compared to the informative ones). Therefore, in sparse networks, the gap between the performance of these methods is much larger than in denser networks. Based on

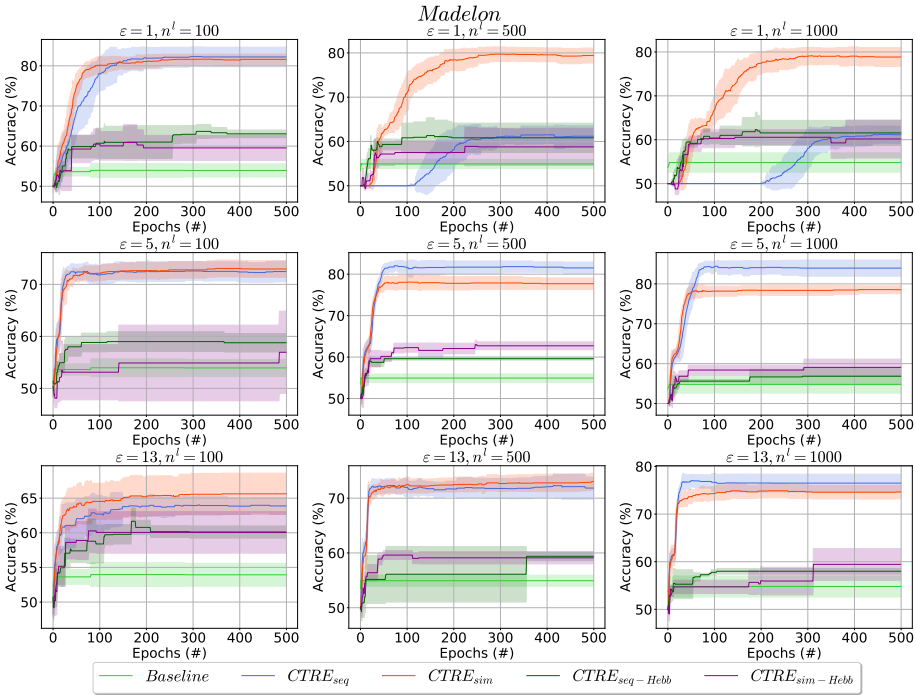


Figure 2.5: Classification accuracy (%) comparison on Madelon for CTRE and pure Hebbian-based updates.

these observations, it can be concluded that the insensitivity of cosine similarity to the vector's magnitude helps CTRE to be more robust in noisy environments.

2.5.4 Convergence Analysis

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

First, we analyze if the weight evolution process in the CTRE algorithm

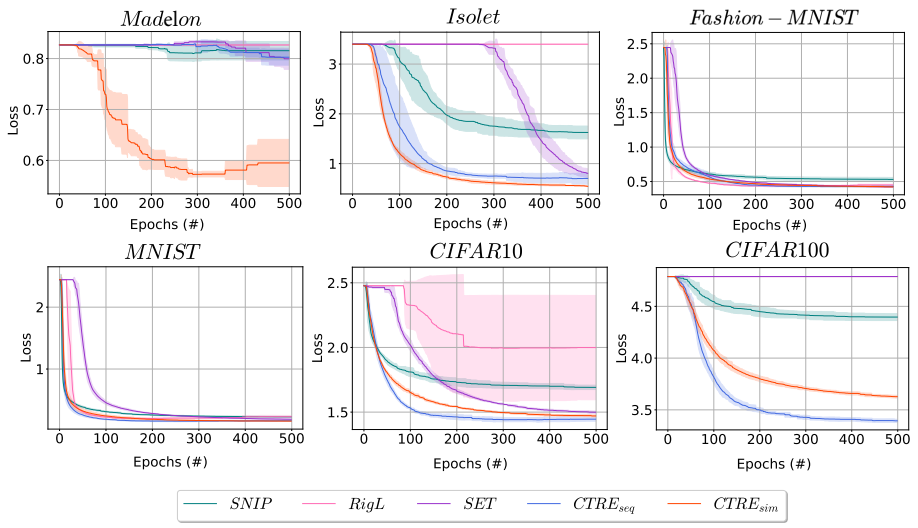


Figure 2.6: Test loss comparison during training for the high sparsity regime and a large network ($\varepsilon = 1$, $n^l = 1000$).

interferes with the convergence of the back-propagation algorithm or not. In the CTRE algorithm, a number of connections are removed at each training epoch, and the same number of connections are added based on the cosine or random search policies. The weight evolution process is performed at each epoch after the standard feed-forward and back-propagation steps. The removed connections have a small magnitude compared to the other connections, and newly activated connections also get a small value. Therefore, they do not change the loss value significantly. The new weights will be updated in the next feed-forward and back-propagation step, and they will grow or shrink. Therefore, the weight evolution process does not disrupt the convergence of the model.

To validate this, we depict the test loss during training in Figure 2.6 for the high sparsity regime and a large network ($\varepsilon = 1$, $n^l = 1000$). It can be observed that the loss function converges for the CTRE algorithm on all the datasets. In addition, in most cases, its convergence speed is much faster than for the other algorithms.

Secondly, we analyze whether CTRE is prone to converge to a local optimum.

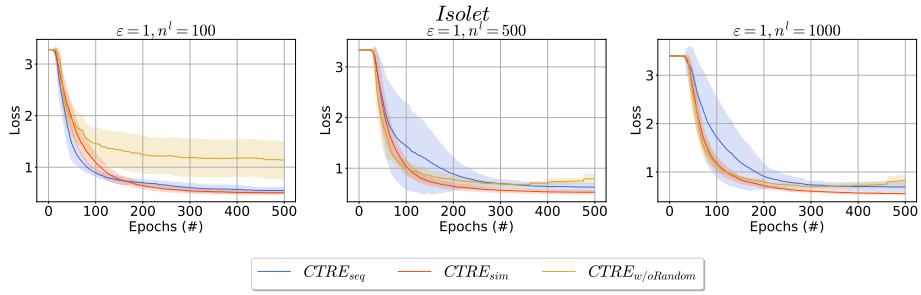


Figure 2.7: Test loss comparison during training for the high sparsity regime ($\varepsilon = 1$) on the Isolet dataset.

As discussed in Section 2.5.2, cosine similarity is very successful at determining the most and least important connections in the network. However, in the mid-importance range, it might not be able to rank connections as well as the magnitude criterion. Therefore, it might add some connections that do not contribute to decreasing the network loss. In such cases, the cosine similarity metric might prevent topology exploration and get stuck in local minima. CTRE explores other weights and exits this local minimum by using a random search. To validate this, in Figure 2.7, we have presented the loss during training for CTRE_{seq}, CTRE_{sim}, and $\text{CTRE}_{\text{w/oRandom}}$ on three highly sparse neural networks trained on the Isolet dataset. The fast decrease in the loss in these plots indicates that all three methods quickly find a good-performing sub-network. However, the loss value of $\text{CTRE}_{\text{w/oRandom}}$ does not improve significantly after 200 epochs, and it converges to a higher value than the other two methods. Therefore, it is important to use random exploration to keep improving the topology and avoid local minima as it is done in CTRE.

2.6 Conclusions and Broader Impacts

In this chapter, we introduced a new biologically plausible sparse training algorithm for finding well-performing highly sparse neural networks, named CTRE. CTRE exploits both the similarity of neurons as an importance metric of the connections and random search, sequentially (CTRE_{seq}) or simultaneously

($CTRE_{sim}$), to explore a performant sparse topology. Our findings indicate that the cosine similarity between neurons' activations can help to evolve a sparse network in a purely sparse manner even in highly sparse scenarios, while most state-of-the-art methods may fail in these cases.

As we demonstrate through extensive experiments CTRE enables a large (having 3 hidden layers with 1000 neurons) and highly sparse (less than 3.4% density level) MLP to reduce the performance gap with the dense counterpart model compared to state-of-the-art methods to obtain sparse neural networks. Therefore, CTRE can be a viable solution for designing Cost-effective Artificial Neural Networks.

By using the neurons' similarity to evolve the topology, our proposed approach can be an excellent initial step toward explainable sparse neural networks. Overall, due to the ability of CTRE to extract highly sparse neural networks, it can be a viable alternative for saving energy in both low-resource devices and data centers and pave the way to achieving environmentally friendly AI systems. Nevertheless, the trade-off between accuracy and sparsity, with CTRE deployed on real-world applications, should be considered carefully. Particularly, if any loss of accuracy may pose safety risks to the user, the sparsity level of the network needs to be analyzed with greater care.

An interesting future direction of this research is to extend CTRE to CNNs; driven by the decent performance of CTRE on image datasets, we believe that it has the potential to be extended to CNN architectures. However, in-depth theoretical analysis and systematic experiments are required to adapt this similarity metric to CNN architectures. This is due to the fact that CNNs require weight sharing, which does not exist in real neurons, and consequently, it is not straightforward to apply Hebbian learning directly [PMLL21]. There have been some efforts to make CNNs more biologically plausible [PMLL21, BSR+18]. Therefore, applying CTRE to CNNs should be done with great care and theoretical analysis.

Despite the remarkable performance of CTRE on tabular and image data, the exploration of sparse neural networks for time series data remains limited. Given the growing volume of large time series datasets generated daily, it is imperative to minimize the computational and memory costs associated with training and deploying neural networks on this type of data. Consequently, one promising approach to tackle this issue is to develop sparse neural networks capable of addressing the unique challenges posed by time series data.

Chapter 3

Adaptive Sparsity Level during Training for Efficient Time Series Forecasting

In the previous chapter, we proposed a solution to reduce the training and deployment costs of neural networks on tabular and image datasets. Despite the promising performance of sparse neural networks in vision and natural processing tasks, there has not been much study on their application for efficient time series analysis. With the growing volume of time series data and the need to forecast millions of them, there is a pressing demand for the development of computationally efficient forecasting models. In this chapter, first, we quantitatively study pruning dense neural networks for time series forecasting. We show that determining the appropriate sparsity level is challenging in this setting due to the heterogeneity in the loss-sparsity tradeoffs across the datasets. Therefore, in the following of this chapter, we propose Pruning with Adaptive Sparsity Level (PALS), to automatically seek a balance between loss and sparsity, all without the need for a predefined sparsity level. PALS puts together sparse training and during-training methods and introduces the novel "expand" mechanism in training sparse neural networks. This

This Chapter is integrally based on: Zahra Atashgahi, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu, *Adaptive Sparsity Level during Training for Efficient Time Series Forecasting with Transformers,* arXiv preprint arXiv:2305.18382 (under review at ECML-PKDD 2024), 2023.

allows the model to dynamically shrink, expand, or remain stable to find a proper sparsity level, bringing a novel perspective to the training of sparse neural networks. We focus on achieving efficiency in transformers known for their excellent time series forecasting performance but high computational cost. Nevertheless, PALS can be applied directly to any neural network architecture. To this aim, we demonstrate its effectiveness also on the DLinear model. Our findings on six benchmark datasets and five state-of-the-art transformer variants show that PALS substantially reduces model size while maintaining comparable performance to the dense model. Our source code is available at https://github.com/zahraatashgahi/PALS.

3.1 Introduction

The capabilities of transformers [VSP+17] for learning long-range dependencies [WDS+20, DBK+20, SRC+21] make them an ideal model for time series processing [WZZ+22]. Several transformer variants have been proposed for the task of time series forecasting, which is crucial for real-world applications, e.g., weather forecasting, energy management, and financial analysis, and have proven to significantly increase the prediction capacity in long time series forecasting (LTSF) [LWWL22]. In addition, attention-based models are inherently an approach for increasing the interpretability for time series analysis in critical applications [LZ21]. Moreover, recent transformer time series forecasting models (e.g., [WXWL21, ZMW+22, LWWL22]) perform generally well in other time series analysis tasks, including, classification, anomaly detection, and imputation [WHL+22].

Despite the outstanding performance of transformers, these models are computationally expensive due to their large model sizes as shown in [SGM20] for natural language processing. With the ever-increasing collection of large time series and the need to forecast millions of them, the requirement to develop computationally efficient forecasting models is becoming significantly critical [THA+18, HLW16, RCM+12]. For industry-scale time series data, which are often high-dimensional and long-length, deploying transformers requires automatically discovering memory- and computationally-efficient architectures that are scalable and practical for real-world applications [WZZ+22]. While there have been some efforts to reduce the computational complexity of transformers in time series forecasting [ZMW+22, ZZP+21], these models have in order of millions of parameters, that can be too large for resource-limited applications, e.g., mobile phones. The over-parameterization of these networks causes high training and inference costs, and their deployment in low-resource environ-

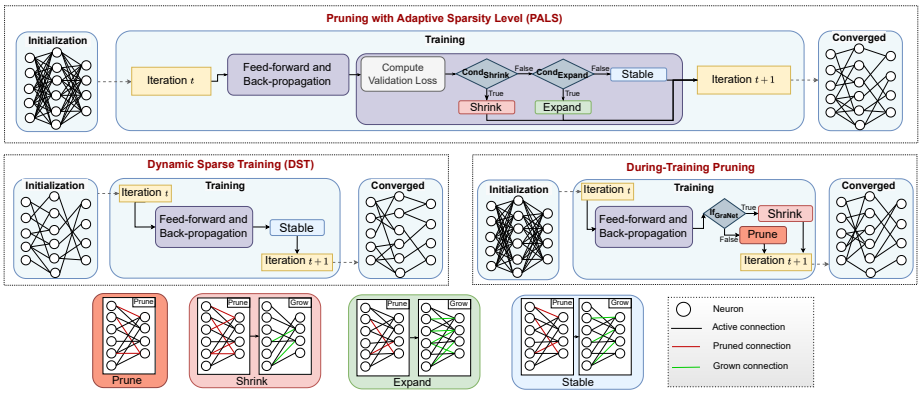


Figure 3.1: Schematic overview of the proposed method, **PALS** (Algorithm 3), Dynamic Sparse Training (DST) [MMS+18, EGM+20], During-training pruning (Gradual Magnitude Pruning (GMP) [ZG17], and GraNet [LCC+21]). While DST and during-training pruning use a fixed sparsity schedule to achieve a predetermined sparsity level at the end of the training, PALS updates the sparse connectivity of the network at each Δt iterations during training, by deciding whether to "Shrink" (decrease density) or "Expand" (increase density) the network or remain "Stable" (same density), to automatically find a proper sparsity level.

ments (e.g., lack of GPUs) would be infeasible. To address these issues, we raise the research question: *How can we reduce the computational and memory overheads of training and deploying transformers for time series forecasting without compromising the model performance?*

Seeking sparsity through sparse connectivity is a widely-used technique to address the over-parameterization of deep learning models [HABN+21]. Early approaches for deriving a sparse sub-network prune a trained dense model [HPTD15], known as *post-training* pruning. While these methods can match the performance of the dense network as shown by the Lottery Ticket Hypothesis[1] (LTH) [FC19], they are computationally expensive during training due to the training of the dense network. *During-training* pruning aims to maintain training efficiency by gradually pruning a dense network during training [LCC+21].

[1] "A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations."

Sparse training [MMS+18] pushed the limits further by starting with a sparse network from scratch and optimizing the topology during training. However, as we study in Section 3.3, the main challenge when using any of these techniques for time series forecasting is to find the proper sparsity level automatically.

In this work, we aim to move beyond optimizing a single objective (e.g. minimizing loss) and investigate sparsity in DNNs for time series prediction in order to find a good trade-off between computational efficiency and performance automatically. Our contributions are:

- We analyze the effect of sparsity (using unstructured pruning) in SOTA transformers for time series prediction [LWWL22, ZMW+22, WXWL21, ZZP+21], and vanilla transformer [VSP+17]. We show they can be pruned up to 80% of their connections in most cases, without significant loss in performance.
- We propose an algorithm, called "**P**runing with **A**daptive **S**parsity **L**evel" (**PALS**) that finds a decent loss-sparsity trade-off by dynamically tuning the sparsity level during training using the loss heuristics and deciding at each connectivity update step weather to *shrink* or *expand* the network, or keep it *stable*. PALS creates a bridge between during-training pruning and dynamic sparse training research areas by inheriting and enhancing some of their most successful mechanisms, while - up to our best knowledge - introducing for the first time into play also the Expand mechanism. Consequently, PALS does not require a desired pre-defined sparsity level which is necessary for most pruning or sparse training algorithms. Figure 3.1 presents the general concept behind PALS.
- We evaluate the performance of PALS in terms of the loss, parameter count, and FLOPs on six widely-used benchmarks for time series prediction and show that PALS can substantially sparsify the models and reduce parameter count and FLOPs. Surprisingly, PALS can even outperform the dense model on average, in 12 and 14 cases out of 30 cases in terms of Mean Squared Error (MSE) and Mean Absolute Error (MAE) loss, respectively (Table 3.3).

3.2 Background

3.2.1 Sparse Neural Networks

Sparse neural networks (SNNs) use sparse connectivity among layers to reduce the computational complexity of DNNs while maintaining a close performance

to the dense counterpart in terms of prediction accuracy. SNNs can be achieved using dense-to-sparse or sparse-to-sparse approaches [MMP+21].

Dense-to-sparse methods prune a dense network; based on the pruning phase, they are categorized into three classes: *post-training* [HPTD15, FC19], *Before-training* [LAT19], and *during-training* [ZG17, LWK18, LCC+21] pruning. Post-training pruning suffers from high computational costs during training and before-training approaches usually fall behind the performance of the dense counter-part network. In contrast, during-training approaches, maintain close or even better performance to the dense network while being efficient through the training process. A standard during-training pruning is Gradual Magnitude Pruning (GMP) [ZG17] which gradually drops unimportant weights based on the magnitude during the training process. GraNet [LCC+21] is another during-training algorithm that gradually shrinks (decreasing density) a network to reach a pre-determined sparsity level. It prunes the weights (as performed in GMP) while allowing for connection regeneration (as seen in Dynamic Sparse Training (DST) which will be explained in the following). As the number of grown weights is less than the pruned ones, the network is shrunk and the density is decreased. For more details regarding GraNet, please refer to Appendix B.2.

Sparse-to-sparse methods start with a random sparse network from scratch and the number of parameters is usually fixed during training and can be determined based on the available computational budget. The sparse topology can remain fixed (static) [MMN+16] or dynamically optimized during training (a.k.a. Dynamic Sparse Training (DST)) [MMS+18, EGM+20, APL+22, JPR+20, LYMP21, YMN+21, SMM+21]. At each topology update iteration, a fraction of unimportant weights are dropped (usually based on magnitude), and the same number of weights are grown. The growth criteria can be random, as in the Sparse Evolutionary Training (SET) [MMS+18], or gradient, as in the Rigged Lottery (RigL) [EGM+20].

Table 3.1: Comparison of related work.

Method	Shrink	Stable	Expand	Adaptive sparsity schedule	Automatically tuning sparsity level
RigL [EGM+20]	✗	✓	✗	✗	✗
GMP [ZG17]	✓	✗	✗	✗	✗
GraNet [LCC+21]	✓	✓	✗	✗	✗
PALS (ours)	✓	✓	✓	✓	✓

In this work, we take advantage of the successful mechanism of "*Shrink*" from during-training pruning (e.g., GraNet [LCC+21]) and "*Stable*" from DST (e.g., RigL [EGM+20]) and propose for the first time the "*Expand*" mechanism, to design a method to automatically optimize the sparsity level during training without requiring to determine it beforehand. Each of these mechanisms is explained in Section 3.4. In Table 3.1, we present a summarized comparison with the closest related work in the literature. Figure 3.1 presents a comprehensive embedding of our proposed method in the literature. Unlike these methods, which update the network using fixed schedules to reach a pre-determined sparsity level, PALS proposes an adaptive approach. It automatically determines whether to shrink or expand the network or remain stable, in order to tune the sparsity level and find a good trade-off between loss and sparsity.

Only a few works investigated SNNs for time series analysis [SHBB22]. [XWZ+22] investigates sparsity in convolutional neural networks (CNNs) for the time series classification and shows their proposed method has superior prediction accuracy while reducing computational costs. [KYGJ19] exploit sparse recurrent neural networks (RNNs) for outlier detection. [LMPP21] and [FST+22] explore sparsity in RNNs for sequence learning.

Sparsity in Transformers

Several works have sought sparsity in transformers [GCL+21, PRR20]. These approaches can be categorized into structured (blocked) [MLN19] or unstructured (fine-grained) pruning [CFC+20]. As discussed in [HABN+21], structured sparsity for transformers is able to only discover models with very low sparsity levels; therefore, we focus on unstructured pruning. [LWS+20] analyses pruning transformers for language modeling tasks and shows that large transformers are robust to compression. [CCG+21] dynamically extract and train sparse subnetworks from Vision Transformers (ViT) [DBK+20] while maintaining a fixed small parameter budget, and they could even improve the accuracy of the ViT in some cases. [DGO+22] investigates DST for BERT language modeling tasks and shows Pareto improvement over the dense model in terms of FLOPs. However, these works mostly focus on vision and NLP tasks. To the best of our knowledge, there is no work that has investigated sparse connectivity in transformers for time series analysis that faces domain-specific challenges as we will elaborate in Section 3.3. Please note that there is a line of research focusing on *sparse attention* [TDBM22] aiming to develop an efficient self-attention mechanism that is orthogonal to our focus in this work (sparsity and pruning) [HABN+21].

3.2.2 Time Series Forecasting

Initial studies for time series forecasting [HA18] exploit classical tools such as ARIMA [BJRL15]. While traditional methods mostly rely on domain expertise or assume temporal dependencies follow specific patterns, machine learning techniques learn the temporal dependencies in a data-driven manner [LZ21, WXZ+22, LLWD22]. In recent years, various deep learning models, including RNNs [WTNM17, QSC+17, SFGJ20], multi-layer perceptrons (MLP) [ZCZX22, ZZC+22], CNNs [LCYL18], and Temporal convolution networks [FDJ19] are utilized to perform time series forecasting [Gam17, OCCB19, COO+22, JPM+22].

Transformers have been extensively used to perform time series forecasting due to their strong ability for sequence modeling. A class of models aims at improving the self-attention mechanism and addresses the computational complexity of vanilla transformers such as LogTrans [LJX+19], Informer [ZZP+21], Reformer [KKL20]. Another category of methods seeks to modify the model to capture the inherent properties of the time series: Autoformer [WXWL21] introduces a seasonal trend decomposition with an auto-correlation block as the attention module. NSTransformer [LWWL22] proposes to add two modules including series stationarization and de-stationary attention in the transformer architecture. FEDformer [ZMW+22] proposes to combine transformers with a seasonal-trend decomposition method to capture the global and detailed behavior of the time series. The research into designing transformers for time series forecasting is ongoing, and many other transformer variants have been proposed, such as TDformer [ZJG+22], Crossformer [ZY23], ETSformer [WLS+22], Pyraformer [LYL+21].

3.2.3 Problem Formulation and Notations

Let $\boldsymbol{x}_t \in \mathbb{R}^m$ denote the observation of a multivariate time series $\boldsymbol{X}$ with m variables at time step t. Given a look-back window $\boldsymbol{X}_{t-L:t} = [\boldsymbol{x}_{t-L}, ..., \boldsymbol{x}_{t-1}]$ of size L, time series forecasting task aims to predict time series over a horizon H as $\widetilde{\boldsymbol{X}}_{t:t+H} = [\widetilde{\boldsymbol{x}}_t, ..., \widetilde{\boldsymbol{x}}_{t+H-1}]$ where $\widetilde{\boldsymbol{x}}_t$ is the prediction at time step t. To achieve this, we need to train a function $f(\boldsymbol{X}_{t-L:t}, \theta)$ (e.g. a transformer network) that can predict future values over horizon H.

In this work, we aim to reduce the model size by pruning the unimportant parameters from θ such that we find the sparse model $f(\boldsymbol{X}_{t-L:t}, \theta_s)$ where $\|\theta_s\|_0 \ll \|\theta\|_0$. $D = \frac{\|\theta_s\|_0}{\|\theta\|_0}$ is called the density level of the model f and $S = 1 - D$ is called as the sparsity level. The aim is to minimize the reconstruction loss between the prediction and the ground truth $\mathcal{L}(f(\boldsymbol{X}_{t-L:t}, \theta_s), \boldsymbol{X}_{t:t+H})$ while finding a proper

sparsity level S automatically. We use Mean Squared Error (MSE) as the loss function such that:

$$\mathcal{L}(\widetilde{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H}) = \frac{1}{H}\Sigma_{i=0}^{H-1}(\widetilde{\boldsymbol{x}}_{t+i} - \boldsymbol{x}_{t+i})^2. \tag{3.1}$$

3.3 Analyzing Sparsity Effect in Transformers for Time Series Forecasting

In this section, we explore sparsity in several time series forecasting transformers. In short, we apply our previous work, GraNet [LCC+21], to prune each model and measure their performance over various sparsity levels.

Experimental Settings

We perform this experiment on six benchmark datasets, presented in Table 3.2. We adapt GraNet [LCC+21], a during-training pruning algorithm developed for CNNs, to sparsify transformer models for time series forecasting. GraNet gradually shrinks a network (here, we start from a dense network) during the training to reach a pre-determined sparsity level, while allowing for connection regeneration inspired by DST. GraNet is described in Appendix B.2. For more details regarding the experimental settings, please refer to Section 3.5.1. For each sparsity level (%) in $\{25, 50, 65, 80, 90, 95\}$, we measure the prediction performance of each transformer model in terms of MSE loss. The results for prediction length $= 96$ (except 24 for the Illness dataset) are presented in Figure 3.2. The results for other prediction lengths are presented in Figure B.1 in Appendix B.2.

Sparsity Effect

We present the results for pruning various transformers in Figure 3.2. It can be observed that most models can be pruned up to 80% or higher sparsity levels without significantly affecting performance. Moreover, a counter-intuitive observation is that in some cases, sparsity does not necessarily lead to worse performance than the dense counterpart, and it can even improve the performance. For example, while on the Electricity, Illness, and Traffic datasets, the behavior is as usually expected (higher sparsity leads to lower performance), on the three other datasets, higher sparsity might even lead to better performance (lower loss) than the dense model. Previous studies have noted instances where sparse models exhibit superior performance compared to dense models. For example,

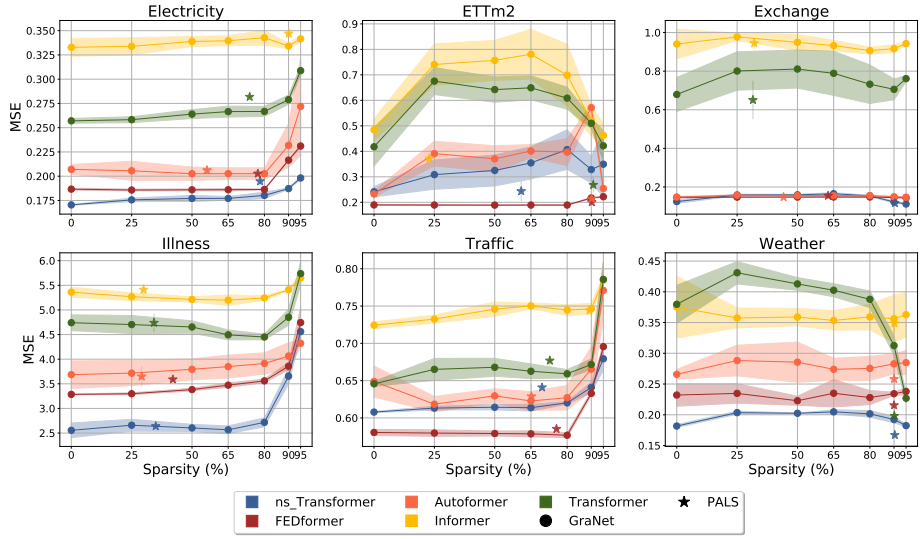


Figure 3.2: Sparsity effect on the performance of various transformer models for time series forecasting on benchmark datasets in terms of MSE loss (prediction length = 96, except 24 for the Illness dataset). Each model is sparsified using GraNet [LCC+21] to sparsity levels (%) $\in \{25, 50, 65, 80, 90, 95\}$ and PALS. $Sparsity = 0$ indicates the original dense model.

in classification with MLPs [MMS+18], feature selection [ASvdL+22], and image classification [LYMP21]. To the best of our knowledge, this phenomenon has not been previously observed in time series analysis tasks. We speculate that this behavior may arise from the overfitting of transformers when applied to straightforward learning tasks; therefore, sparsity improves generalization in such cases. Nevertheless, we let the investigation of this phenomenon to future research endeavors. In addition, the sparsity effect is different among various models, particularly on the latter group of datasets, including the ETTm2, Exchange, and Weather datasets. We discuss the potential reasons for different behavior among datasets in Appendix B.8. Last but not least, by looking at Figure B.1 in Appendix B.2, the prediction length can also be a contributing factor to the sparsity-loss trade-off.

Challenge

Based on the above observations, we can conclude that the sparsity effect is not homogeneous across various time series datasets, forecasting models, and prediction lengths for time series forecasting. Our findings in these experiments are not aligned with the statements in [HABN+21] for CNNs (vision) and Transformers (NLP), where for a given task and technique, increasing the sparsity level results in decreasing the prediction performance. However, we observe in Figures 3.2 and B.1 that increasing the sparsity level does not necessarily lead to decreased performance and it might even significantly improve the performance (e.g. for the vanilla transformer on the Weather dataset). Therefore, it is challenging to decide how much we can push the sparsity level and what is the decent sparsity level without having prior knowledge of the data, model, and experimental settings. While GraNet is the closest in spirit to our proposed method, it cannot automatically tune the sparsity level since it needs the initial and the final sparsity level as its hyperparameters. In this work, we aim to address this challenge by proposing an algorithm that can automatically tune the sparsity level during training.

3.4 Proposed Methodology: PALS

This section presents our proposed method for automatically finding a proper sparsity level of a DNN, called "**P**runing with **A**daptive **S**parsity **L**evel" (**PALS**) (Algorithm 3). While our main focus in this chapter is to sparsify transformer models, PALS is not specifically designed for transformers and can be applied directly to other artificial neural network architectures (See Appendix B.6 for experiments on training with PALS the DLinear [ZCZX22]) model.

3.4.1 Motivation and Broad Outline

As we discussed in Section 3.3, the main challenge when seeking sparsity for time series forecasting is to find a good sparsity level automatically. Therefore, PALS aims to tune the sparsity level during training without requiring prior information about models or datasets. PALS is in essence inspired by the DST framework [MMS+18] and gradual magnitude pruning (GMP) [ZG17, LCC+21]. While DST and GMP use fixed sparsification policies (fixed sparsity level (*Stable* in Figure 3.1) and constantly prune the network until the desired sparsity level is reached (*Shrink* in Figure 3.1), respectively) and require the final sparsity

Algorithm 3 PALS

Input: Time series $\boldsymbol{X} \in \mathbb{R}^{T \times m}$, number of training iterations t_{max}, Sequence length L, Prediction length H, model dimension d_{model}, pruning rate ζ, mask update frequency Δt, Initial density D_{init}, pruning rate factor $\gamma > 1$ and loss freedom factor $\lambda > 1$, sparsity bound S_{min} and S_{max}.
Initialization: Initialize the transformer model with density level D_{init}, $S = 1 - D_{init}$, $L_{best} = \text{inf}$.
Training:
for $t \in \{1, \ldots, \#t_{max}\}$ **do**
    I. Standard feed-forward and back-propagation. The network is trained on $batch_t$ of samples.
    II. Update sparsity mask
    if $(t \mod \Delta t) = 0$ **then**
        Compute Validation Loss L^t_{valid}
        if $(S < S_{min})$ **or** $(L^t_{valid} <= \lambda * L_{best}$ **and** $S < S_{max})$ **then**
            update_mask $(\zeta_{prune} = \gamma * \zeta, \zeta_{grow} = \zeta)$
        else if $L^t_{valid} > \lambda * L_{best}$ **and** $S > S_{best}$ **then**
            update_mask $(\zeta_{prune} = \zeta, \zeta_{grow} = \gamma * \zeta)$
        else
            update_mask $(\zeta_{prune} = \zeta, \zeta_{grow} = \zeta)$
        if $L^t_{valid} < L_{best}$ **then**
            $L_{best} = L^t_{valid}$, $S_{best} = S$
        Set S to the current sparsity level of the network.

level before training, PALS exploits heuristic information from the network at each iteration to automatically determine whether to *increase*, *decrease*, or *keep* the sparsity level at each connectivity update step. While existing growing methods [HPN+17,MQS+22] grow a network or a layer of it to dense connectivity, to the best of our knowledge, this is the first work that allows the network to *expand* by increasing the density during training without requiring dense connectivity, and allows for automatic shrink or expand. If the training starts from a dense neural network ($D_{init} = 1$) PALS can be seen as a dense-to-sparse method, while if $D_{init} < 1$ then PALS is a sparse-to-sparse method.

3.4.2 Training

The training of PALS (Algorithm 3) starts with initializing a network with density level $D_{init} = 1 - S_{init}$. Then, the training procedure of PALS consists of two steps:

1. Standard feed-forward and back-propagation. The network's parameters are updated each training iteration t using a batch of samples.

2. Update Sparse Connectivity. The novelty of the method lies in updating the sparse connectivity. At every Δt iteration, the connectivity is updated in two steps. (2-1) The validation loss at step t is calculated as L^t_{valid}. (2-2) The sparsity mask is updated ($update\_mask$ in Algorithm 3) by first pruning ζ_{prune} of weights with the lowest magnitude:

$$\widetilde{\boldsymbol{W}_l} = Update(\boldsymbol{W}_l, top(|\boldsymbol{W}_l|, 1-\zeta_{prune})), \tag{3.2}$$

where $\boldsymbol{W}_l$ is the l^{th} weight matrix of the network, $Update(\boldsymbol{A}, idx)$ keeps only the indices idx of the matrix $\boldsymbol{A}$, $top(\boldsymbol{A},\zeta)$ returns the indices of a fraction ζ of the largest elements of $\boldsymbol{A}$. Then, we grow ζ_{grow} of the weights with the highest gradients:

$$\boldsymbol{W}_l = \widetilde{\boldsymbol{W}_l} + top(|\boldsymbol{G}_{l,i\notin\widetilde{\boldsymbol{W}_l}}|, \zeta_{grow}) \tag{3.3}$$

where $\boldsymbol{G}_{l,i\notin\widetilde{\boldsymbol{W}_l}}$ is the gradient of zero weights in layer l. These new connections are initialized with zero values. This process is repeated for each layer in the model. Based on the values of ζ_{prune} and ζ_{grow}, PALS determines whether to decrease (*shrink*), increase (*expand*), or keep (*stable*) the density of the network:

- $\mathbf{S_t > S_{t-1}}$ **(Shrink).** If the loss does not go beyond $\lambda \times L_{best}$, we decrease the overall number of parameters such that $\zeta_{prune} = \gamma \times \zeta, \zeta_{grow} = \zeta$. The loss freedom coefficient, $\lambda > 1$, is a hyperparameter of the network that determines how much the loss value can deviate from the best validation loss achieved so far L_{best} during training. The lower λ is, the more strict PALS becomes at allowing the network to go to the *shrink* phase, finally resulting in a lower sparsity network. $\gamma > 1$ is the pruning factor coefficient, which determines how much to prune or grow more in the shrink and expand phases, respectively. We analyze the sensitivity of PALS to λ and γ in Section 3.6.2. In addition, we define a boundary for sparsity determined by S_{min} and S_{max} which can be determined by the user based on the available resources. If the sparsity level does not meet the minimum sparsity level S_{min}, we prune the network more than we grow. If the network sparsity goes beyond S_{max}, we do not increase sparsity.

- $\mathbf{S_t < S_{t-1}}$ **(Expand).** If $S > S_{best}$ (S_{best} is the sparsity level corresponding to L_{best}) and the loss goes higher than $\lambda \times L_{best}$, it means that the earlier pruning step(s) were not beneficial to decreasing the loss (improving

forecasting quality in the time series forecasting) and the network requires a higher capacity to recover a good performance. Therefore, we expand the network and grow more connections than the pruned ones at this step: $\zeta_{prune} = \zeta, \zeta_{grow} = \gamma \times \zeta$.

- $\mathbf{S_t = S_{t-1}}$ **(Stable).** If none of the above cases happened, we only update a fraction ζ of the network's parameters without changing the sparsity level: $\zeta_{prune} = \zeta, \zeta_{grow} = \zeta$.

  For a better understanding of how the sparsity level evolves during the training process of PALS, please refer to Appendix B.7.4.

3.5 Experiments and Results

In this section, we evaluate PALS on several transformers for time series forecasting.

3.5.1 Experimental Settings

Datasets

The experiments are performed on six widely-used benchmark datasets for time series forecasting. The datasets are summarized in Table 3.2 and described in Appendix B.1. These datasets have different characteristics including stationary and non-stationary with/without obvious periodicity.

Datasets in Table 3.2 have been carefully selected to encompass a diverse array of characteristics. Such characteristics include: 1) *Sampling frequency*: while the sampling frequency for some time series datasets is very high (e.g., Electricity (1 hour) and Weather datasets (10 minutes)), it might be very low for some others (Exchange (1 day) and Illness (1 week)) 2) *Periodicity of the*

Table 3.2: Datasets characteristics.

Dataset	# Variables	Sampling Frequency	# Observations
Electricity	321	1 Hour	26304
ETTm2	7	15 Miutes	69680
Exchange	8	1 Day	7588
Illness	7	1 Week	966
Traffic	862	1 Hour	17544
Weather	21	10 Minutes	52695

variables: time series datasets can be periodic (ETTm2) or without obvious periodicity (Exchange) 3) *Number of variables*: the number of variables can vary significantly. Some datasets have below 10 variables (ETT, Illness) while others have in order of hundreds (Traffic, Electricity). This characteristic results in different levels of complexity. Therefore, automatically tuning the sparsity can help to tune the complexity of the task at hand eventually helping prevent over-fitting in simple tasks (e.g., Weather) and maintaining over-parameterization for complex tasks (e.g., Traffic, Electricity). The beauty of our proposed method consists in the fact that it does not have to consider any of these intrinsic differences. We did not make any fine-tuning for PALS to account for these differences, and it does everything automatically. Of course, fine-tuning PALS per dataset specificity would improve its final performance, but it would reduce the generality of our proposed work and we prefer not to do it.

Models

We consider five SOTA transformer models for time series forecasting, including Non-Stationary Transformer (NSTransformer) [LWWL22], FEDformer [ZMW+22], Autoformer [WXWL21], Informer [ZZP+21], and vanilla transformer [VSP+17]. Please refer to Section 3.2.2 for more details.

It is important to acknowledge that other time series forecasting models can serve as baselines for comparison. In our evaluations, we specifically included the state-of-the-art (SOTA) time series forecasting model to measure the performance of our proposed approach.

Evaluation metrics

We evaluate the methods in two aspects: 1) Quality of the prediction in terms of MSE and MAE, and 2) Computational complexity in terms of parameter count and FLOPs (Floating-point operations). We report the theoretical FLOPs to be independent of the used hardware, as it is done in the unstructured pruning literature [LCC+21, EGM+20]. A lower value for these metrics indicates higher prediction quality and lower computational complexity, respectively. We measure the performance of each model for various prediction lengths $H \in \{96, 192, 336, 720\}$ (except $H \in \{24, 36, 48, 60\}$ for the Illness dataset).

Implementation

Experiments are implemented in PyTorch. The start of implementation is the NSTransformer[2] and GraNet[3]. We repeat each experiment for three random seeds and report the average of the runs. In the experiments, D_{init} was set to 1, thus PALS can be seen as a during-training pruning method. We have run the experiments on *Intel Xeon Platinum 8360Y CPU* and one *NVIDIA A100 GPU*. We will discuss the hyperparameters' settings in Appendix B.1.

3.5.2 Results

Multivariate Time Series Prediction

The results in terms of MSE and parameter count for the considered datasets and models are presented in Table B.1 in Appendix B.3. In most cases considered, PALS decreases the model size by more than 50% without a significant increase in loss. More interestingly, in most cases on the ETTm2, Exchange, and Weather datasets PALS even achieves lower MSE than the dense counterpart.

To summarize the results of Table B.1 (Appendix B.3) and have a general overview of the performance of PALS on each model and dataset, we present the average MSE and MAE, and parameters count in addition to the difference between the dense and the sparse model using PALS (in percentage) in Table 3.3. Additionally, we include the inference FLOPs count (total FLOPs for all test samples). It can be observed that PALS even outperforms the dense model in 12 and 14 cases out of 30 cases in terms of MSE and MAE loss, respectively, while reducing 65% parameter count and 63% FLOPs on average. We summarize the training FLOPs in Appendix B.7.1.

Based on the experiments conducted in Section 3.3 and the description of datasets provided in Appendix B.1.1, we observed significant variations in the sparsity-loss trade-off across different datasets and models. The beauty of our proposed method consists in the fact that it does not have to consider any of these differences. We did not make any finetuning for PALS to account for these differences, and it does everything automatically. Of course, finetuning PALS per dataset and model specificity would improve its final performance, but it would reduce the generality of our proposed work and we prefer not to do it.

[2] *https://github.com/thuml/Nonstationary_Transformers*
[3] *https://github.com/VITA-Group/GraNet*

Table 3.3: Summary of the results on the benchmark Datasets in Table B.1. For each experiment on a transformer model and dataset, the average MSE, MAE, number of parameters ($\times 10^6$), and the inference FLOPs count ($\times 10^{12}$) for various prediction lengths are reported before and after applying PALS. The difference between these results is shown in % where the blue color means improvement of PALS compared to the corresponding dense model.

Model	Electricity				ETTm2				Exchange			
	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs
NSTransformer	**0.19**	**0.30**	12.0	9.25	0.49	0.43	10.6	19.82	0.54	0.49	10.6	1.89
+PALS	0.21	0.32	2.2	1.81	0.38	0.39	2.5	3.70	**0.49**	**0.47**	**5.4**	**1.07**
Difference	10.8% ↑	7.3% ↑	81.5% ↓	80.5% ↓	24.0% ↓	11.2% ↓	76.7% ↓	81.3% ↓	9.3% ↓	3.6% ↓	48.5% ↓	43.3% ↓
FEDformer	0.21	0.32	19.5	9.30	**0.30**	0.35	17.9	19.82	0.50	0.49	17.9	1.89
+PALS	0.23	0.34	3.0	1.35	0.30	**0.35**	1.8	1.96	0.51	0.50	10.5	1.15
Difference	9.0% ↑	4.9% ↑	84.7% ↓	85.5% ↓	1.5% ↑	0.5% ↓	90.2% ↓	90.1% ↓	2.1% ↑	1.1% ↑	41.2% ↓	38.9% ↓
Autoformer	0.24	0.34	12.1	9.30	0.33	0.37	10.5	19.82	0.58	0.53	10.5	1.89
+PALS	0.26	0.36	2.7	1.71	0.31	0.35	**1.0**	**1.93**	0.62	0.55	7.1	1.30
Difference	9.3% ↑	4.6% ↑	77.7% ↓	81.6% ↓	8.1% ↓	5.6% ↓	90.3% ↓	90.3% ↓	5.5% ↑	4.4% ↑	32.7% ↓	31.0% ↓
Informer	0.36	0.43	12.5	8.51	1.53	0.88	11.3	18.15	1.59	1.00	11.3	1.71
+PALS	0.42	0.48	**1.4**	**0.94**	1.39	0.83	5.3	8.45	1.53	0.98	8.6	1.33
Difference	18.9% ↑	11.3% ↑	88.6% ↓	88.9% ↓	9.0% ↓	5.8% ↓	53.4% ↓	53.4% ↓	3.9% ↓	1.6% ↓	24.2% ↓	22.4% ↓
Transformer	0.28	0.38	11.7	9.24	1.48	0.86	10.5	19.81	1.61	0.97	10.5	1.89
+PALS	0.31	0.40	2.5	2.24	1.08	0.75	3.2	8.17	1.41	0.91	6.6	1.16
Difference	10.5% ↑	5.7% ↑	78.3% ↓	75.8% ↓	26.7% ↓	13.3% ↓	69.9% ↓	58.8% ↓	12.5% ↓	6.1% ↓	37.5% ↓	38.4% ↓
$\text{Difference}_{\text{avg}}$	11.7% ↑	6.8% ↑	82.1% ↓	82.5% ↓	13.3% ↓	7.3% ↓	76.1% ↓	74.8% ↓	3.6% ↓	1.2% ↓	36.8% ↓	34.8% ↓
Model	Illness				Traffic				Weather			
	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs
NSTransformer	**2.14**	**0.92**	10.5	0.05	0.63	**0.34**	14.2	6.61	0.29	0.31	10.7	18.10
+PALS	2.33	0.97	7.4	0.04	0.67	0.37	4.3	2.05	**0.26**	**0.29**	1.0	1.77
Difference	9.1% ↑	5.1% ↑	30.0% ↓	30.2% ↓	5.2% ↑	9.1% ↑	70.1% ↓	69.0% ↓	10.2% ↓	6.9% ↓	90.3% ↓	90.2% ↓
FEDformer	2.84	1.14	13.7	0.05	**0.61**	0.38	22.3	6.71	0.32	0.37	17.9	18.11
+PALS	3.05	1.19	8.3	0.03	0.62	0.38	5.6	1.83	0.31	0.36	1.8	1.81
Difference	7.2% ↑	5.0% ↑	39.5% ↓	39.6% ↓	1.0% ↑	1.1% ↑	74.6% ↓	72.7% ↓	2.8% ↓	3.2% ↓	90.0% ↓	90.0% ↓
Autoformer	3.08	1.18	10.5	0.05	0.64	0.40	14.9	6.71	0.34	0.38	10.6	18.11
+PALS	3.19	1.22	**6.7**	0.03	0.65	0.41	4.5	1.94	0.34	0.38	1.3	2.52
Difference	3.5% ↑	3.2% ↑	36.6% ↓	36.5% ↓	1.9% ↑	2.1% ↑	69.5% ↓	71.0% ↓	0.1% ↑	1.3% ↓	87.7% ↓	86.1% ↓
Informer	5.27	1.58	11.3	0.05	0.81	0.46	14.4	6.14	0.62	0.55	11.4	16.58
+PALS	5.23	1.57	7.4	**0.03**	0.94	0.53	**2.3**	**1.19**	0.69	0.56	4.3	8.35
Difference	0.8% ↓	1.0% ↓	34.9% ↓	34.8% ↓	15.5% ↑	15.3% ↑	83.9% ↓	80.6% ↓	12.1% ↑	2.0% ↑	61.9% ↓	49.7% ↓
Transformer	4.94	1.49	10.5	0.05	0.67	0.36	13.6	6.61	0.64	0.56	10.6	18.10
+PALS	4.91	1.48	7.7	0.04	0.69	0.38	3.8	1.86	0.32	0.38	**1.0**	**1.76**
Difference	0.7% ↓	0.9% ↓	27.2% ↓	27.3% ↓	3.3% ↑	5.4% ↑	71.7% ↓	71.8% ↓	49.6% ↓	32.5% ↓	90.2% ↓	90.3% ↓
$\text{Difference}_{\text{avg}}$	3.7% ↑	2.3% ↑	33.6% ↓	33.7% ↓	5.4% ↑	6.6% ↑	74.0% ↓	73.0% ↓	10.1% ↓	9.0% ↓	84.0% ↓	81.3% ↓

Univariate Time Series Prediction

The results of univariate prediction (using a single variable) on the ETTm2 and Exchange datasets are presented in Table B.2 and summarized in Table B.3 in Appendix B.4. In short, PALS outperforms the dense counterpart model on average, in 7 and 8 cases out of 12 cases in terms of MSE and MAE loss, respectively.

3.6 Discussion

In this section, we study the performance of PALS in comparison with other pruning and DST algorithms (3.6.1) and the hyperparameter sensitivity of PALS (3.6.2). Additionally in the Appendix, we analyze the performance of PALS in terms of model size (B.8), prediction quality by visualizing the predictions (B.9), pruning DLinear [ZCZX22] (B.6), and computational efficiency from various aspects (B.7).

3.6.1 Performance Comparison with Pruning and Sparse Training Algorithms

We compare PALS with a standard during-training pruning approach (GMP [ZG17]), GraNet [LCC+21], and a well-known DST method (RigL [EGM+20]). These are the closest methods in the literature in terms of including gradual pruning and gradient-based weight regrowth.

While PALS derives a proper sparsity level automatically, other pruning approaches require the sparsity level as an input of the algorithm. Therefore, to compare PALS with existing pruning algorithms, the sparsity level should be optimized for them. We apply GraNet, RigL, and GMP to NSTransformer for prediction lengths of $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$). For each of these methods (GraNet, RigL, and GMP), the sparsity level is optimized among values of $\{25, 50, 65, 80, 90, 95\}$. This means that for one run of PALS, we run the other methods 6 times. The model with the lowest validation loss is used to report the test loss. Table 3.4 summarizes the average loss, sparsity level, and training epochs (due to early stopping the algorithms might not require the full training) over different prediction lengths.

The closest competitor of PALS is GraNet. In Table 3.4, for the Electricity dataset, PALS achieves a sparsity level of 80.5% with a loss of 0.21, while GraNet achieves a sparsity level of only 31.2% with a slightly lower loss of 0.20. Similarly,

for the ETTm2 dataset, PALS achieves a sparsity level of 76.7% with a loss of 0.38, while GraNet achieves a higher sparsity level of 95.0% but with a much higher loss of 0.60. On the other datasets, they perform relatively close to each other.

By looking at the results of all methods in Table 3.4, PALS has the highest average sparsity value (66.0%) compared to GraNet (63.73%), RigL (48.3%), and GMP (62.4%). While RigL requires fewer training epochs (~ 6.6 epochs) compared to PALS (~ 7.2 epochs), it finds lower sparsity networks and has a higher average loss (RigL: 0.74 compared to PALS: 0.72). GraNet and GMP use fixed pruning schedules, and as a result, they need almost full training time (~ 9.5 epochs). The only extra computational requirement of PALS compared to GraNet is an additional step that involves determining the number of weights to prune and grow. This is negligible when considering the overall computation necessary for training the models. On the other hand, as PALS does not require the full training epochs in contrast to GraNet, it needs much lower computational costs. We additionally compared the convergence speed of PALS with the dense model in Appendix B.7.3.

In short, PALS has the lowest average loss and highest sparsity values compared to other algorithms, suggesting that PALS could build efficient and accurate sparse neural networks for time series forecasting.

Table 3.4: Comparison with other during-training pruning methods (GMP, GraNet) and a DST method (RigL) when sparsifying NSTransformer. The results are average over four prediction lengths.

	PALS			**GraNet***			**RigL***			**GMP***		
Dataset	**l**	**S**	**e**	**l**	**S**	**e**	**l**	**S**	**e**	**l**	**S**	**e**
Electricity	0.21	80.5%	8.83	0.20	31.2%	9.75	0.20	31.2%	9.12	0.20	47.5%	9.62
ETTm2	0.38	76.7%	4.58	0.60	95.0%	9.00	0.49	77.5%	4.33	0.60	56.2%	9.12
Exchange	0.49	48.5%	5.83	0.47	95.0%	9.42	0.44	90.0%	4.25	0.45	95.0%	9.50
Illness	2.33	30.0%	7.97	2.32	25.0%	9.58	2.37	25.0%	9.58	2.22	31.2%	9.92
Traffic	0.67	70.1%	8.83	0.64	41.2%	9.50	0.64	25.0%	8.17	0.64	50.0%	9.79
Weather	0.26	90.3%	7.00	0.28	95.0%	9.08	0.27	41.2%	4.08	0.29	95.0%	9.08
Average	**0.72**	**66.0%**	7.17	0.75	63.73%	9.38	0.74	48.3%	**6.60**	0.73	62.4%	9.47

* Optimized sparsity level (%) in {25, 65, 50, 80, 90, 95}. GraNet, RigL, and GMP, each require 6 runs to optimize the sparsity level while PALS needs only one run.

3.6.2 Hyperparameter Sensitivity

In this section, we discuss the sensitivity of PALS to its hyperparameters including pruning rate factor γ and loss freedom factor λ. We have changed their values in $\{1.05, 1.1, 1.2\}$ and measured the performance of PALS (with NSTransformer) in terms of MSE and parameter count on six benchmark datasets. The results are presented in Table B.4 in Appendix B.5.

As shown in Table B.4, PALS is not very sensitive to its hyperparameters and the results in each row are close in terms of loss in most cases considered. However, by increasing γ and λ PALS tends to find a sparser model. A small λ results in paying more attention to the loss value, while a large value gives more freedom to PALS to explore a sparse sub-network that might sometimes result in a higher loss value. A small γ limits the amount of additional grow/prune in the expand/shrink phase, while a large γ gives more flexibility to the algorithm for exploring various sparsity levels. In short, a small value for each of these hyperparameters makes PALS more strict and allows for small changes in sparse connectivity, while a large value increases the exploration rate which potentially results in higher sparsity and/or reduced loss.

3.7 Conclusions

In this chapter, we showed that pruning neural networks for time series forecasting can be challenging in terms of determining the proper sparsity level. This is due to the heterogeneous loss-sparsity trade-off for various datasets, prediction lengths, and models.

With the aim of decreasing the computational and memory costs of training and deploying DNNs for time series forecasting, we proposed PALS. PALS is a novel method to obtain sparse neural networks, that exploits loss heuristics to automatically find the best trade-off between loss and sparsity in one round of training. PALS leverages the effective strategies of "Shrink" from during-training pruning and "Stable" from DST. Additionally, we introduce a novel strategy called the "Expand" mechanism. The latter allows PALS to automatically optimize the sparsity level during training, eliminating the need for prior determination.

We assessed the effectiveness of PALS in terms of prediction performance and computational efficiency through extensive experimental evaluation. We showed that PALS is able to (1) find a decent trade-off between loss and sparsity without a predefined sparsity level, (2) outperform dense training in several cases in terms of prediction performance and computational efficiency, (3) outperform DST and

pruning competitors in terms of convergence speed and the loss-sparsity trade-off (4) be independent from the underlying forecasting model and performing well on transformers (complex model) and MLP-based model (simple model).

An open direction to this research can be to start with a highly sparse neural network (as opposed to starting from a dense network used in PALS) and gradually expand the network to be even more efficient during training.

Up to this point in the thesis, we have introduced two solutions aimed at mitigating the expenses associated with the over-parameterization of neural networks during both their training and deployment phases. These solutions have demonstrated their applicability across a wide range of data types, including tabular, image, and time series data. Nevertheless, a significant lingering challenge pertains to the costs incurred due to high-dimensional data. It is imperative to be mindful of these expenses when seeking for a cost-effective model.

Chapter 4

Quick and Robust Feature Selection

Although the costs associated with the over-parametrization of neural networks can be substantial, another critical bottleneck in reducing the training and deployment costs of these models pertains to the input data. Input to neural networks can be high-dimensional which can result in increased size and the consequent computational and memory overhead. Moreover, high-dimensional feature sets frequently include noisy and irrelevant features that can decrease generalization and slow down training convergence. While feature selection has historically been a remedy, most existing feature selection methods are computationally inefficient, resulting in increased energy consumption, an undesirable outcome for devices with constrained computational and energy resources. In this chapter, we propose for the first time to leverage sparse neural networks to perform feature selection and introduce an energy-efficient method for unsupervised feature selection, named QuickSelection. QuickSelection exploits the strength of the neuron in sparse neural networks as a criterion to measure the feature importance. This criterion, blended with sparsely connected denoising autoencoders trained with the sparse evolutionary training procedure, derives the importance of all input features simultaneously. We implement QuickSelection in a purely sparse manner as opposed

---

This Chapter is integrally based on: Zahra Atashgahi, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy, *Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders,* **Machine Learning**, **ECML-PKDD 2022 journal track**, Vol. 111, pp. 377–414, (2022).

to the typical approach of using a binary mask over connections to simulate sparsity. This design substantially accelerates processing speed and reduces memory usage. When tested on several benchmark datasets, including five low-dimensional and three high-dimensional datasets, the proposed method is able to achieve the best trade-off of classification and clustering accuracy, running time, and maximum memory usage, among widely used approaches for feature selection. Besides, our proposed method requires the least amount of energy among the state-of-the-art autoencoder-based feature selection methods. The source code is available at: https://github.com/zahraatashgahi/QuickSelection

4.1 Introduction

In the last few years, considerable attention has been paid to the problem of dimensionality reduction and many approaches have been proposed [VDMPVdH09]. There are two main techniques for reducing the number of features of a high-dimensional dataset: feature extraction and feature selection. Feature extraction focuses on transforming the data into a lower-dimensional space. This transformation is done through a mapping which results in a new set of features [LM98]. Feature selection reduces the feature space by selecting a subset of the original attributes without generating new features [CS14]. Based on the availability of the labels, feature selection methods are divided into three categories: supervised [AMHH15, CS14], semi-supervised [ZL07, SSGC17], and unsupervised [MN16, DB04]. Supervised feature selection algorithms try to maximize some function of predictive accuracy given the class labels. In unsupervised learning, the search for discriminative features is done blindly, without having the class labels. Therefore, unsupervised feature selection is considered a much harder problem [DB04].

Feature selection methods improve the scalability of machine learning algorithms since they reduce the dimensionality of data. Besides, they reduce the ever-increasing demands for computational and memory resources that are introduced by the emergence of big data. This can lead to a considerable decrease in energy consumption in data centers. This can ease not only the problem of high energy costs in data centers but also the critical challenges imposed on the environment [YXJ+18]. As outlined by the High-Level Expert Group on Artificial Intelligence (AI) [Gro20], environmental well-being is one of the requirements of a trustworthy AI system. The development, deployment, and process of an AI system should be assessed to ensure that it would function in the most environmentally friendly way possible. For example, resource usage

and energy consumption through training can be evaluated.

However, a challenging problem that arises in the feature selection domain is that selecting features from datasets that contain a huge number of features and samples, may require a massive amount of memory, computational, and energy resources. Since most of the existing feature selection techniques were designed to process small-scale data, their efficiency can be downgraded with high-dimensional data [BCSMAB15]. Only a few studies have focused on designing feature selection algorithms that are efficient in terms of computation [TTW14, ASL+18]. The main contributions of this chapter can be summarized as follows:

- We propose a new fast and robust unsupervised feature selection method, named QuickSelection. As briefly sketched in Figure 4.1, It has two key components: (1) Inspired by node strength in graph theory, the method proposes the neuron strength of sparse neural networks as a criterion to measure the feature importance; and (2) The method introduces sparsely connected Denoising Autoencoders (sparse DAEs) trained from scratch with the sparse evolutionary training procedure to model the data distribution efficiently. The imposed sparsity before training also reduces the amount of required memory and the training running time.
- We implement QuickSelection in a completely sparse manner in Python using the SciPy library and Cython rather than using a binary mask over connections to simulate sparsity. This ensures minimum resource requirements, i.e., just Random-Access Memory (RAM) and a Central Processing Unit (CPU), without demanding a Graphic Processing Unit (GPU).

The experiments performed on 8 benchmark datasets suggest that QuickSelection has several advantages over the state-of-the-art, as follows:

- It is the first or the second-best performer in terms of both classification and clustering accuracy in almost all scenarios considered.
- It is the best performer in terms of the trade-off between classification and clustering accuracy, running time, and memory requirement.
- The proposed sparse architecture for feature selection has at least one order of magnitude fewer parameters than its dense equivalent. This leads to the outstanding fact that the wall clock training time of QuickSelection running on the CPU is smaller than the wall clock training time of its autoencoder-based competitors running on GPU in most cases.

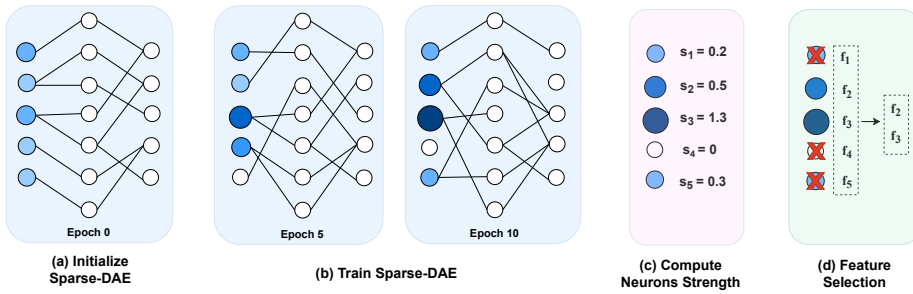


Figure 4.1: A high-level overview of the proposed method, "QuickSelection". (a) At epoch 0, connections are randomly initialized. (b) After initializing the sparse structure, we start the training procedure. After 5 epochs, some connections are changed during the training procedure, and as a result, the strength of some neurons has increased or decreased. At epoch 10, the network has converged, and we can observe which neurons are important (larger and darker blue circles) and which are not. (c) When the network is converged, we compute the strength of all input neurons. (d) Finally, we select K features corresponding to neurons with the highest strength values.

- Last but not least, QuickSelection's computational efficiency makes it have the minimum energy consumption among the autoencoder-based feature selection methods considered.

4.2 Related Work

4.2.1 Feature Selection

The literature on feature selection shows a variety of approaches that can be divided into three major categories, including filter, wrapper, and embedded methods. Filter methods use a ranking criterion to score the features and then remove the features with scores below a threshold. These criteria can be Laplacian score [HCN06], Correlation, Mutual Information [CS14], and many other scoring methods such as Bayesian scoring function, t-test scoring, and Information theory-based criteria [LTM+12]. These methods are usually fast and computationally efficient. Wrapper methods evaluate different subsets of features to detect the best subset. Wrapper methods usually give better performance than

filter methods; they use a predictive model to score each subset of features. However, this results in high computation complexity. Seminal contributions for this type of feature selection have been made by [KJ97]. In [KJ97], the authors used a tree structure to evaluate the subsets of features. Embedded methods unify the learning process, and the feature selection [LCWE06]. Multi-Cluster Feature Selection (MCFS) [CZH10] is an unsupervised method for embedded feature selection, which selects features using spectral regression with L1-norm regularization. A key limitation of this algorithm is that it is computationally intensive since it depends on computing the eigenvectors of the data similarity matrix and then solving an L1-regularized regression problem for each eigenvector [FGK13]. Unsupervised Discriminative Feature Selection (UDFS) [YSM+11] is another unsupervised embedded feature selection algorithm that simultaneously utilizes both feature and discriminative information to select features [LCW+18].

A closely related body of research concentrates on applying evolutionary algorithms for feature selection in reinforcement learning settings. FS-NEAT [WSS+05] introduces an extension to the NEAT evolutionary algorithm and performs automatic feature selection using a model-free approach. Follow-up studies have used this evolutionary approach to perform model-based feature selection [KW09] or in online settings [BM13]. Notably, our methodology differs from these approaches as we undertake feature selection within unsupervised learning paradigms, in contrast to the reinforcement learning settings explored in these studies. Furthermore, our approach is only loosely inspired by evolutionary algorithms. Due to the expansive search space involved, the direct application of such algorithms to our problem would entail a notable increase in computational time.

4.2.2 Autoencoders for Feature Selection

In the last few years, many deep learning-based models have been developed to select features from the input data using the learning procedure of deep neural networks [LCW16]. In [LFLN18], a Multi-Layer Perceptron (MLP) is augmented with a pairwise-coupling layer to feed each input feature along with its knockoff counterpart into the network. After the training, the authors use the filter weights of the pairwise-coupling layer to rank input features. Autoencoders which are generally known as a strong tool for feature extraction [BCV13], are being explored to perform unsupervised feature selection. In [HWZ+18], authors combine autoencoder regression and group lasso task for unsupervised feature selection named Autoencoder Feature Selector (AEFS). In [DS19], an autoencoder is combined with three variants of structural regularization to

perform unsupervised feature selection. These regularizations are based on slack variables, weights, and gradients, respectively. Another recently proposed autoencoder-based embedded method is feature selection with Concrete Autoencoder (CAE) [BAZ19]. This method selects features by learning a concrete distribution over input features. They proposed a concrete selector layer that selects a linear combination of input features that converges to a discrete set of K features during training. In [SY20], the authors showed that a large set of parameters in CAE might lead to over-fitting in case of having a limited number of samples. In addition, CAE may select features more than once since there is no interaction between the neurons of the selector layer. To mitigate these problems, they proposed a concrete neural network feature selection (FsNet) method, which includes a selector layer and a supervised deep neural network. The training procedure of FsNet considers reducing the reconstruction loss and maximizing the classification accuracy simultaneously. In our research, we focus mostly on unsupervised feature selection methods.

Denoising Autoencoder (DAE) is introduced to solve the problem of learning the identity function in the autoencoders. This problem is most likely to happen when we have more hidden neurons than inputs [Bal12]. As a result, the network output may be equal to the inputs, which makes the autoencoder useless. DAEs solve the aforementioned problem by introducing noise to the input data and trying to reconstruct the original input from its noisy version [VLBM08]. As a result, DAEs learn a representation of the input data that is robust to small irrelevant changes in the input. In this research, we use the ability of this type of neural network to encode the input data distribution and select the most important features. Moreover, we demonstrate the effect of noise addition on the feature selection results.

4.2.3 Sparse Training

Deep neural networks usually have at least some fully connected layers, which results in a large number of parameters. In a high-dimensional space, this is not desirable since it may cause a significant decrease in training speed and a rise in memory requirement. To tackle this problem, sparse neural networks have been proposed. Pruning the dense neural networks is one of the most well-known methods to achieve a sparse neural network [LDS90,HS93]. [HPTD15] start from a pre-trained network, prune the unimportant weights, and retrain the network. Although this method can output a network with the desired sparsity level, the minimum computation cost is as much as the cost of training a dense network. To reduce this cost, [LAT19] start with a dense neural network, and prune it prior

to training based on connection sensitivity. Then, the sparse network is trained in the standard way. However, starting from a dense neural network requires at least the memory size of the dense neural network and the computational resources for one training iteration of a dense network. Therefore, this method might not be suitable for low-resource devices.

In 2016, [MMN+16] introduced the idea of training sparse neural networks from scratch, a concept that recently has started to be known as sparse training. The sparse connectivity pattern was fixed before training using graph theory, network science, and data statistics. While it showed promising results, outperforming the dense counterpart, the static sparsity pattern did not always model the data optimally. In order to address these issues, in 2018, [MMS+18] proposed the Sparse Evolutionary Training (SET) algorithm which makes use of dynamic sparsity during training. The idea is to start with a sparse neural network before training and dynamically change its connections during training to automatically model the data distribution. This results in a significant decrease in the number of parameters and increased performance. SET evolves the sparse connections at each training epoch by removing a fraction ζ connections with the smallest magnitude, and randomly adding new connections in each layer. [BPR+19] have shown that a sparse MLP trained with SET achieves state-of-the-art results on tabular data in predicting human decisions, outperforming fully-connected neural networks and Random Forest, among others.

In this work, we introduce for the first time sparse training in the world of denoising autoencoders, and we named the newly introduced model sparse denoising autoencoder (sparse DAE). We train the sparse DAE with the SET algorithm to keep the number of parameters low, during the training. Then, we exploit the trained network to select the most important features.

4.3 Proposed Method

To address the problem of the high dimensionality of the data, we propose a novel method, named "QuickSelection," to select the most informative attributes from the data, based on their strength (importance). In short, we train a sparse denoising autoencoder network from scratch in an unsupervised adaptive manner. Then, we use the trained network to derive the strength of each neuron in the input features.

The basic idea of our proposed approach is to impose sparse connections on DAE, which proved its success in the related field of feature extraction, to efficiently handle the computational complexity of high-dimensional data in

terms of memory resources. Sparse connections are evolved in an adaptive manner that helps in identifying informative features.

A couple of methods have been proposed for training deep neural networks from scratch using sparse connections and sparse training [DZ19, MMS+18, BKML18, MW19, EGM+20, ZJ19]. All these methods are implemented using a binary mask over connections to simulate sparsity since all standard deep learning libraries and hardware (e.g. GPUs) are not optimized for sparse weight matrix operations. Unlike the aforementioned methods, we implement our proposed method in a purely sparse manner to meet our goal of actually using the advantages of a very small number of parameters during training. We decided to use SET in training our sparse DAE.

The choice of SET is due to its desirable characteristics. SET is a simple method yet achieves satisfactory performance. Unlike other methods that calculate and store information for all the network weights, including the non-existing ones, SET is memory efficient. It stores the weights for the existing sparse connections only. It does not need any high computational complexity as the evolution procedure depends on the magnitude of the existing connections only. This is a favorable advantage to our proposed method to select informative features quickly. In the following subsections, we first present the structure of our proposed sparse denoising autoencoder network and then explain the feature selection method. The pseudo-code of our proposed method can be found in Algorithm 4.

4.3.1 Sparse DAE

Structure. As the goal of our proposed method is to do fast feature selection in a memory-efficient way, we consider here the model with the least possible number of hidden layers, one hidden layer, as more layers mean more computation. Initially, sparse connections between two consecutive layers of neurons are initialized with an Erdős–Rényi random graph which has been shown to be more effective than uniform (equal sparsity between layers) distribution [EGM+20, GMB23]. In Erdős–Rényi distribution, the probability of the connection between two neurons is given by

$$P(W_{ij}^{l}) = \frac{\epsilon(n^{l-1} + n^{l})}{n^{l-1} \times n^{l}}, \tag{4.1}$$

where ϵ denotes the parameter that controls the sparsity level, n^{l} denotes number of neurons at layer l, and W_{ij}^{l} is the connection between neuron i in layer $l-1$

and neuron j in layer l, stored in the sparse weight matrix $\mathbf{W}^l$.

Input denoising. We use the additive noise model to corrupt the original data:

$$\tilde{\mathbf{x}} = \mathbf{x} + nf\mathcal{N}(\mu, \sigma^2), \tag{4.2}$$

where $\mathbf{x}$ is the input data vector from dataset X, nf (noise factor) is a hyperparameter of the model which determines the level of corruption, and $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian noise. After denoising the data, we derive the hidden representation $\mathbf{h}$ using this corrupted input. Then, the output $\mathbf{z}$ is reconstructed from the hidden representation. Formally, the hidden representation $\mathbf{h}$ and the output $\mathbf{z}$ are computed as follows:

$$\mathbf{h} = a(\mathbf{W}^1\tilde{\mathbf{x}} + \mathbf{b}^1), \tag{4.3}$$

$$\mathbf{z} = a(\mathbf{W}^2\mathbf{h} + \mathbf{b}^2), \tag{4.4}$$

where $\mathbf{W}^1$ and $\mathbf{W}^2$ are the sparse weight matrices of hidden and output layers respectively, $\mathbf{b}^1$ and $\mathbf{b}^2$ are the bias vectors of their corresponding layer, and a is the activation function of each layer. The objective of our network is to reconstruct the original features in the output. For this reason, we use mean squared error (MSE) as the loss function to measure the difference between original features $\mathbf{x}$ and the reconstructed output $\mathbf{z}$:

$$L_{MSE} = \|\mathbf{z} - \mathbf{x}\|_2^2. \tag{4.5}$$

Finally, the weights can be optimized using the standard training algorithms (e.g., Stochastic Gradient Descent (SGD), AdaGrad, and Adam) with the above reconstruction error.

Training procedure. We adapt the SET training procedure [MMS+18] in training our proposed network for feature selection. SET works as follows. After each training epoch, a fraction ζ of the smallest positive weights and a fraction ζ of the largest negative weights at each layer is removed. The selection is based on the magnitude of the weights. New connections in the same amount as the removed ones are randomly added in each layer. Therefore the total number of connections in each layer remains the same, while the number of connections per neuron varies, as represented in Figure 4.1. The weights of these new connections are initialized from a standard normal distribution. The random

addition of new connections does not have a high risk of not finding a good sparse connectivity at the end of the training process because it has been shown in [LvdLY+20] that sparse training can unveil a vast number of very different sparse connectivity local optima which achieve very similar performance.

4.3.2 Feature Selection

We select the most important features of the data based on the weights of their corresponding input neurons of the trained sparse DAE. Inspired by node strength in graph theory [BBPSV04], we determine the importance of each neuron based on its **strength**. We estimate the strength of each neuron (s_i) by the summation of absolute weights of its outgoing connections.

$$s_i = \sum_{j=1}^{n^1} |W_{ij}^1|, \tag{4.6}$$

where n^1 is the number of neurons of the first hidden layer, and W_{ij}^1 denotes the weight of connection linking input neuron i to hidden neuron j.

Algorithm 4 QuickSelection

Input: Dataset X, noise factor nf, sparsity hyperparameter ϵ, number of hidden neurons n^h, number of selected features K
Input denoising: $\tilde{\mathbf{x}} = \mathbf{x} + nf\mathcal{N}(\mu, \sigma^2)$
Structure initialization: Initialize sparse-DAE with n^h hidden neurons and sparsity level determined by ϵ
procedure TRAINING SPARSE-DAE
    Let the loss be $L_{MSE} = \|\mathbf{z} - \mathbf{x}\|_2^2$ where z is the output of sparse-DAE
    for $i \in \{1, \ldots, epochs\}$ **do**
        Perform standard forward propagation and backpropagation
        Perform weight removal and addition for topology optimization
procedure QUICKSELECTION
    Compute neurons strength:
    for $i \in \{1, \ldots, \#inputfeatures\}$ **do**
        $s_i = \sum_{j=1}^{n^h} |W_{ij}^1|$
    Select K features: $\mathbb{F}_s^* = \arg\max_{\mathbb{F}_s \subset \mathbb{F}, |\mathbb{F}_s| = K} \sum_{f_i \in \mathbb{F}_s} s_i$,

As represented in Figure 4.1, the strength of the input neurons changes during training. We have depicted the strength of the neurons according to their size and color. After convergence, we compute the strength of all of the input neurons; each input neuron corresponds to a feature. Then, we select the features corresponding to the neurons with K largest strength values:

$$\mathbb{F}_s^* = \underset{\mathbb{F}_s \subset \mathbb{F}, |\mathbb{F}_s| = k}{\arg\max} \sum_{f_i \in \mathbb{F}_s} s_i, \tag{4.7}$$

where $\mathbb{F}$ and $\mathbb{F}_s^*$ are the original feature set and the final selected features respectively, f_i is the i^{th} feature of $\mathbb{F}$, and K is the number of features to be selected. In addition, by sorting all the features based on their strength, we will derive the importance of all features in the dataset. In short, we will be able to rank all input features by training just once a single sparse DAE model.

For a deeper understanding of the above process, we analyze the strength of each input neuron in a 2D map on the MNIST dataset. As illustrated in Figure 4.2, at the beginning of training, all the neurons have small strength due to the random initialization of each weight to a small value. During the network evolution, stronger connections are linked to important features gradually. We can observe that after ten epochs, the neurons in the center of the map become stronger. This pattern is similar to the pattern of MNIST data in which most of the digits appear in the middle of the picture.

We studied other metrics for estimating the neuron importance such as the strength of output neurons, degree of input and output neurons, and strength and degree of neurons simultaneously. However, in our experiments, all these methods have been outperformed by the strength of the input neurons in terms of accuracy and stability.

4.4 Experiments

In order to verify the validity of our proposed method, we carry out several experiments. In this section, first, we state the settings of the experiments, including hyperparameters and datasets. Then, we perform feature selection with QuickSelection and compare the results with other methods, including MCFS, Laplacian Score, and three autoencoder-based feature selection methods. After that, we do different analyses on QuickSelection to understand its behavior. Finally, we discuss the scalability of QuickSelection and compare it with the other methods considered.

4.4.1 Settings

The experiment settings, including the values of hyperparameters, implementation details, the structure of the sparse DAE, datasets we use for evaluation, and the evaluation metric, are as follows.

Hyperparameters and Implementation

For feature selection, we consider the case of the simplest sparse DAE with one hidden layer consisting of 1000 neurons. This choice is made due to our main objective to decrease the model complexity and the number of parameters. The activation function used for the hidden and output layer neurons is "Sigmoid" and "Linear" respectively, except for the Madelon dataset where we use "Tanh" for the output activation function. We train the network with SGD and a learning rate of 0.01. The hyperparameter ζ, the fraction of weights to be removed in the SET procedure, is 0.2. Also, ϵ, which determines the sparsity level, is set to 13. We set the noise factor (nf) to 0.2 in the experiments. To improve the learning process of our network, we standardize the features of our dataset such that each attribute has zero mean and unit variance. However, for SMK and PCMAC datasets, we use Min-Max scaling. The preprocessing method for each dataset is determined with a small experiment of the two preprocessing methods.

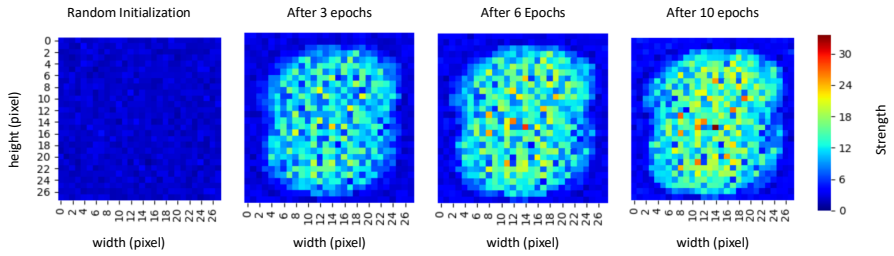


Figure 4.2: Neuron's strength on the MNIST dataset. The heat maps above are a 2D representation of the input neuron's strength. It can be observed that the strength of neurons is random at the beginning of training. After a few epochs, the pattern changes, and neurons in the center become more important and similar to the MNIST data pattern.

We implement sparse DAE and QuickSelection[1] in a purely sparse manner in Python, using the Scipy library [JOP01] and Cython. We compare our proposed method to MCFS, Laplacian score (LS), AEFS, and CAE, which have been mentioned in Section 4.2. We also performed some experiments with UDFS; however, since we were not able to obtain many of the results within the considered time limit (24 hours), we did not include the results for the comparison. We have used the scikit-feature repository for the implementation of MCFS, and Laplacian score [LCW+18]. Also, we use the implementation of feature selection with CAE and AEFS from Github[2]. In addition, to highlight the advantages of using sparse layers, we compare our results with a fully-connected autoencoder (FCAE) using the neuron strength as a measure of the importance of each feature. To have a fair comparison, the structure of this network is considered similar to our DAE, one hidden layer containing 1000 neurons implemented using TensorFlow. Furthermore, we have studied the effect of other components of QuickSelection, including input denoising and SET training algorithm, in Appendix C.2.1 and C.6, respectively.

For all the other methods (except FCAE for which all the hyperparameters and preprocessing are similar to QuickSelection), we scaled the data between zero and one, since it yields better performance than data standardization for these methods. The hyperparameters of the aforementioned methods have been set similarly to the ones reported in the corresponding code or paper. For AEFS, we tuned the regularization hyperparameter between 0.0001 and 1000, since this method is sensitive to this value. We perform our experiments on a single CPU core, Intel Xeon Processor E5 v4, and for the methods that require GPU, we use NVIDIA TESLA P100.

Datasets

We evaluate the performance of our proposed method on eight datasets, including five low-dimensional datasets and three high-dimensional ones. Table 4.1 illustrates the characteristics of these datasets.

- **COIL-20** [NNM+96] consists of 1440 images taken from 20 objects (72 poses for each object).

[1]The implementation of QuickSelection is available at: *https://github.com/zahraatashgahi/QuickSelection*

[2]The implementation of AEFS and CAE is available at: *https://github.com/mfbalin/Concrete-Autoencoders*

Table 4.1: Datasets characteristics.

Dataset	Dimensions	Type	Samples	Train	Test	Classes
Coil20	1024	Image	1440	1152	288	20
Isolet	617	Speech	7737	6237	1560	26
HAR	561	Time Series	10299	7352	2947	6
Madelon	500	Artificial	2600	2000	600	2
MNIST	784	Image	70000	60000	10000	10
SMK-CAN-187	19993	Microarray	187	149	38	2
GLA-BRA-180	49151	Microarray	180	144	36	4
PCMAC	3289	Text	1943	1554	389	2

- **Madelon** [GGNZ08] is an artificial dataset with 5 informative features and 15 linear combinations of them. The rest of the features are distractor features since they have no predictive power.
- **Human Activity Recognition** (HAR) [AGO+13] is created by collecting the observations of 30 subjects performing 6 activities such as walking, standing, and sitting. The data was recorded by a smartphone connected to the subjects' body.
- **Isolet** [FC91] has been created with the spoken name of each letter of the English alphabet.
- **MNIST** [LeC98] is a database of 28x28 images of handwritten digits.
- **SMK-CAN-187** [SBS+07] is a gene expression dataset with 19993 features. This dataset compares smokers with and without lung cancer.
- **GLA-BRA-180** [SHS+06] consists of the expression profile of Stem cell factors useful to determine tumor angiogenesis.
- **PCMAC** [Lan95] is a subset of the 20 Newsgroups data.

We select diverse data types for evaluating our proposed algorithm. We believe this is essential for several reasons. Firstly, different data types exhibit unique characteristics, such as dimensionality, noise levels, and inherent structures, which can profoundly impact the performance and effectiveness of feature selection techniques. By assessing the algorithm's performance across varied data types, we gain insights into its robustness and adaptability to different scenarios. Secondly, real-world datasets often encompass a wide range of domains, including text, image, audio, and tabular data, among others. A feature

selection algorithm validated solely on one type of data may not generalize well to others, potentially limiting its practical utility. Therefore, evaluating the algorithm across diverse data types enables us to validate its efficacy across multiple application domains, ensuring broader applicability and reliability in real-world settings. Lastly, exploring diverse data types fosters a deeper understanding of how the algorithm operates across different modalities, facilitating insights into its strengths, weaknesses, and potential areas for improvement. Overall, selecting different data types for evaluation provides us with a comprehensive assessment of a feature selection algorithm's performance, generalizability, and suitability for real-world applications.

The datasets selected within each category type are frequently utilized in prior feature selection studies. This deliberate choice facilitates our comparisons with earlier works, as we leverage datasets that have already been well-established within the field.

Evaluation Metrics

To evaluate our model, we compute two metrics: clustering accuracy and classification accuracy. To derive clustering accuracy [LCW+18], first, we perform K-means using the subset of the dataset corresponding to the selected features and get the cluster labels. Then, we find the best match between the class labels and the cluster labels and report the clustering accuracy. We repeat the K-means algorithm 10 times and report the average clustering results since K-means may converge to a local optimal.

To compute classification accuracy, we use a supervised classification model named "Extremely randomized trees" (ExtraTrees), which is an ensemble learning method that fits several randomized decision trees on different parts of the data [GEW06]. The choice of the classification method is made due to the computational efficiency of the ExtraTrees classifier. To compute classification accuracy, first, we derive the K selected features using each feature selection method considered. Then, we train the ExtraTrees classifier with 50 trees as estimators on the K selected features of the training set. Finally, we compute the classification accuracy on the unseen test data. For the datasets that do not contain a test set, we split the data into training and testing sets, including 80% of the total original samples for the training set and the remaining 20% for the testing set. In addition, we have evaluated the classification accuracy of feature selection using the random forest classifier [LW+02] in Appendix C.7.

Table 4.2: Clustering accuracy (%) using 50 selected features (except Madelon for which we select 20 features). On each dataset, the bold entry is the best performer, and the italic one is the second-best performer.

Method	COIL-20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
MCFS	**67.0±0.7**	*33.8±0.5*	**62.4±0.0**	57.2±0.0	35.2±0	51.6±0.2	**65.8±0.3**	50.6±0.0
LS	55.5±0.4	33.2±0.2	*61.2±0.0*	*58.1±0.0*	14.9±0.1	51.6±0.4	55.5±0.4	50.6±0.0
CAE	60.0±1.1	31.6±1.3	51.4±0.4	56.9±3.6	**49.2±1.5**	**60.7±0.4**	55.4±1.3	*52.0±1.2*
AEFS	51.2±1.7	31.0±2.7	55.0±2.2	50.8±0.2	40.0±1.9	52.4±1.8	56.1±5.2	50.9±0.5
FCAE	*60.2±1.7*	28.7±2.5	49.5±8.7	50.9±0.4	28.2±8.5	51.5±0.8	53.5±3.0	50.9±0.1
QS_{10}	59.5±2.1	32.5±2.8	56.0±2.6	57.5±3.8	45.4±3.9	*54.0±3.1*	53.6±4.7	50.9±0.5
QS_{100}	*60.2±2.0*	**35.1±2.7**	54.6±4.5	**58.2±1.5**	*48.3±2.4*	51.8±0.8	*59.5±1.8*	**52.5±1.1**
QS_{best}	63.8±1.5	42.2±2.6	59.5±4.3	58.6±0.9	48.3 ±2.4	54.9±1.39	59.5±1.8	53.1±0

4.4.2 Feature Selection

We set the hyperparameter for the number of the features to select to 50 for all datasets except Madelon, for which we set this number to just 20 features since most of its features are non-informative noise. Then, we compute the clustering and classification accuracy on the selected subset of features; the more informative features selected, the higher the accuracy will be achieved. The clustering and classification accuracy results of our model and the other methods are summarized in Tables 4.2 and 4.3, respectively. These results are an average of 5 runs for each case. For the autoencoder-based feature selection methods, including CAE, AEFS, and FCAE, we consider 100 training epochs. However, we present the results of QuickSelection at epochs 10 and 100 named $\text{QuickSelection}_{10}$ and $\text{QuickSelection}_{100}$, respectively. This is mainly due to the fact that our proposed method is able to achieve a reasonable accuracy after the first few epochs. Moreover, we perform hyperparameter tuning for ϵ and ζ using the grid search method over a limited number of values for all datasets; the best result is presented in Table 4.2 and 4.3 as $\text{QuickSelection}_{\text{best}}$. The results of hyperparameter selection can be found in Appendix C.7.1. However, we do not perform hyperparameter optimization for the other methods (except AEFS). Therefore, in order to have a fair comparison between all methods, we do not compare the results of $\text{QuickSelection}_{\text{best}}$ with the other methods.

From Table 4.2, it can be observed that QuickSelection outperforms all the other methods on Isolet, Madelon, and PCMAC, in terms of clustering accuracy, while being the second-best performer on Coil20, MNIST, SMK, and

Table 4.3: Classification accuracy (%) using 50 selected features (except Madelon for which we select 20 features). On each dataset, the bold entry is the best-performer, and the italic one is the second-best performer.

Method	COIL-20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
MCFS	99.2±0.3	79.5±0.4	88.9±0.3	81.7±0.8	88.7±0	75.8±1.5	70.6±3.8	55.5±0.0
LS	89.8±0.4	83.0±0.2	86.4±0.4	**91.4±0.9**	20.7±0.1	71.6±5.6	71.7±1.1	50.4±0.0
CAE	*99.6±0.3*	**89.8±0.6**	**91.7±1.0**	87.5±2.0	**95.4±0.1**	71.6±3.1	70.0±4.1	**59.9±1.5**
AEFS	93.0±2.7	85.1±2.4	87.7±1.4	52.1±2.8	86.1±2.0	*76.3±4.4*	68.9±3.7	57.1±3.6
FCAE	**99.7±0.2**	81.6±5.9	87.4±2.4	53.5±8.1	68.8±28.7	71.6±3.5	*72.8±4.8*	58.1±1.9
QS_{10}	98.8±0.6	86.9±1.1	88.8±0.7	86.6±3.6	*93.8±0.6*	**76.9±4.6**	69.4±3.0	*58.9±4.4*
QS_{100}	**99.7±0.3**	*89.0±1.3*	*90.2±1.2*	*90.3±0.7*	93.5±0.5	75.7±3.9	**73.3±3.3**	58.0±2.9
QS_{best}	99.7±0.3	89.0±1.3	90.5±1.6	90.9 ±0.5	94.2±0.5	81.6 ±2.9	73.3±3.3	61.3±6.1

GLA. Furthermore, On the HAR dataset, it is the best performer among all the autoencoder-based feature selection methods considered. As shown in Table 4.3, QuickSelection outperforms all the other methods on Coil20, SMK, and GLA, in terms of classification accuracy, while being the second-best performer on the other datasets. From these Tables, it is clear that QuickSelection can outperform its equivalent dense network (FCAE) in terms of classification and clustering accuracy on all datasets.

It can be observed in Tables 4.2 and 4.3, that Lap_score has a poor performance when the number of samples is large (e.g. MNIST). However, in the tasks with a low number of samples and features, even in noisy environments such as Madelon, Lap_score has a relatively good performance. In contrast, CAE has a poor performance in noisy environments (e.g., Madelon), while it has a decent classification accuracy on the other datasets considered. It is the best or second-best performer on five datasets, in terms of classification accuracy, when $K = 50$. AEFS and FCAE cannot achieve a good performance on Madelon, either. We believe that the dense layers are the main cause of this behavior; the dense connections try to learn all input features, even the noisy features. Therefore, they fail to detect the most important attributes of the data. MCFS performs decently on most of the datasets in terms of clustering accuracy. This is due to the main objective of MCFS to preserve the multi-cluster structure of the data. However, this method also has poor performance on datasets with a large number of samples (e.g., MNIST) and noisy features (e.g., Madelon).

However, since evaluating the methods using a single value of K might not be

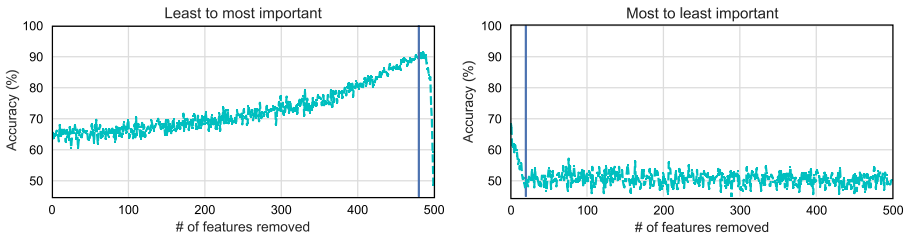


Figure 4.3: Influence of feature removal on Madelon dataset. After deriving the importance of the features with QuickSelection, we sort and then remove them based on the above two methods.

enough for comparison, we performed another experiment using different values of K. In Appendix C.1.1, we test other values for K on all datasets and compare the methods in terms of classification accuracy, clustering accuracy, running time, and maximum memory usage. The summary of the results of this Appendix has been summarized in Section 4.5.1.

Relevancy of Selected Features

To illustrate the ability of QuickSelection to find informative features, we analyze thoroughly the Madelon dataset results, which have the interesting property of containing many noisy features. We perform the following experiments; first, we sort the features based on their strength. Then, we remove the features one by one from the least important feature to the most important one. In each step, we train an ExtraTrees classifier with the remained features. We repeat this experiment by removing the features from the most important ones to the least important ones. The result of classification accuracy for both experiments can be seen in Figure 4.3. On the left side of Figure 4.3, we can observe that removing the least important features, which are noise, increases the accuracy. The maximum accuracy occurs after we remove 480 noise features. This corresponds to the moment when all the noise features are supposed to be removed. In Figure 4.3 (right), it can be seen that removing the features in a reverse order results in a sudden decrease in the classification accuracy. After removing 20 features (indicated by the vertical blue line), the classifier performs like a random classifier. We conclude that QuickSelection is able to find the most informative features in good order.

To better show the relevancy of the features found by QuickSelection, we

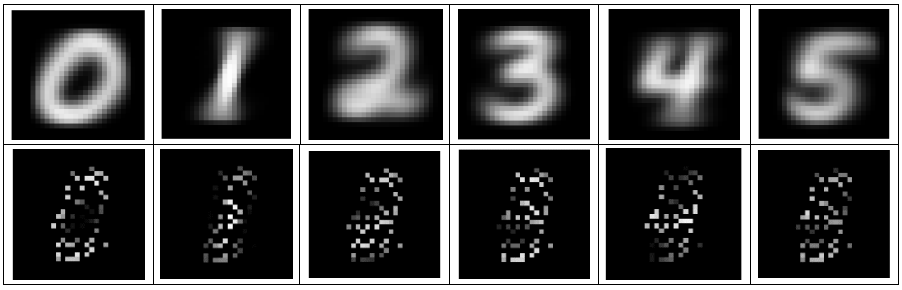


Figure 4.4: Average values of all data samples of each class corresponding to the 50 selected features on MNIST after 100 training epochs (bottom), along with the average of the actual data samples of each class (top).

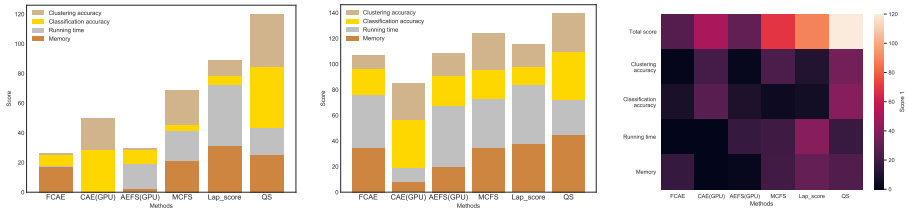
visualize the 50 features selected on the MNIST dataset per class, by averaging their corresponding values from all data samples belonging to one class. As can be observed in Figure 4.4, the resulting shape resembles the actual samples of the corresponding digit. We discuss the results of all classes at different training epochs in more detail in Appendix C.3.

4.5 Discussion

4.5.1 Accuracy and Computational Efficiency Trade-off

In this section, we perform a thorough comparison between the models in terms of running time, energy consumption, memory requirement, clustering accuracy, and classification accuracy. In short, we change the number of features to be selected (K) and measure the accuracy, running time, and maximum memory usage across all methods. Then, we compute two scores to summarize the results and compare methods.

We analyze the effect of changing K on QuickSelection performance and compare it with other methods; the results are presented in Figure C.1 in Appendix C.1.1. Figure C.1a compares the performance of all methods when K is changing between 5 and 100 on low-dimensional datasets, including Coil20, Isolet, HAR, and Madelon. Figure C.1b illustrates performance comparison for K between 5 and 300 on the MNIST dataset, which is also a low-dimensional dataset. We discuss this dataset separately since it has a large number of samples that make



(a) Score 1. Scores are given based on the ranking of the methods.

(b) Score 2. Scores are given based on the normalized value of each objective.

(c) Heat-map visualization of Score 1.

Figure 4.5: Feature selection comparison in terms of classification accuracy, clustering accuracy, speed, and memory requirement, on each dataset and for different values of K, using two scoring variants.

it different from other low-dimensional datasets. Figure C.1c represents a similar comparison on three high-dimensional datasets, including SMK, GLA, and PCMAC. It should be noted that to have a fair comparison, we use a single CPU core to run these methods; however, since the implementations of CAE and AEFS are optimized for parallel computation, we use a GPU to run these methods. We also measure the running time of feature selection with CAE on CPU.

To compare the memory requirement of each method, we profile the maximum memory usage during feature selection for different values of K. The results are presented in Figure C.2 in Appendix C.1.1, derived using a Python library named resource[3]. Besides, to compare the memory occupied by the autoencoder-based models, we count the number of parameters for each model. The results are shown in Figure C.5 in the Appendix C.1.3.

However, comparing all of these methods only by looking into the graphs in Figure C.1 and Figure C.2 is not easily possible, and the trade-off between the factors is not clear. For this reason, we compute two scores to take all these metrics into account simultaneously.

Score 1. To compute this score, on each dataset and for each value of K, we rank the methods based on the running time, memory requirement, clustering accuracy, and classification accuracy. Then, we give a score of 1 to the best and second-best performers; this is mainly due to the fact that in most cases, the difference between these two is negligible. After that, we compute the summation of these scores for each method on all datasets. The results are

[3]https://docs.python.org/2/library/resource.html

presented in Figure 4.5a; to ease the comparison of different components in the score, a heat-map visualization of the scores is presented in Figure 4.5c. The cumulative score for each method consists of four parts that correspond to each metric considered. As it is obvious in this Figure, QuickSelection (cumulative score of QuickSelection$_{10}$ and QuickSelection$_{100}$) outperforms all other methods by a significant gap. Our proposed method is able to achieve the best trade-off between accuracy, running time, and memory usage, among all these methods. Laplacian score, the second-best performer, has a decent performance in terms of running time and memory, while it cannot perform well in terms of accuracy. On the other hand, CAE has a satisfactory performance in terms of accuracy. However, it is not among the best two performers in terms of computational resources for any values of K. Finally, FCAE and AEFS cannot achieve a decent performance compared to the other methods. A more detailed version of Figure 4.5a is available in Figure C.3 in Appendix C.1.1.

Score 2. In addition to the ranking-based score, we calculate another score to consider all the methods, even the lower-ranking ones. With this aim, on each dataset and value of K, we normalize each performance metric between 0 and 1, using the values of the best performer and worst performer on each metric. The value of 1 in the accuracy score means the highest accuracy. However, for the memory and running time, the value of 1 means the least memory requirement and the least running time, respectively. After normalizing the metrics, we accumulate the normalized values for each method and on all datasets. The results are depicted in Figure 4.5b. As can be seen in this diagram, QuickSelection (we consider the results of QuickSelection$_{100}$) outperforms the other methods by a large margin. CAE has a close performance to QuickSelection in terms of both accuracy metrics, while it has a poor performance in terms of memory and running time. In contrast, Lap_score is computationally efficient while having the lowest accuracy score. In summary, it can be observed in Figure 4.5b, that QuickSelection achieves the best trade-off of the four objectives among the considered methods.

Energy Consumption. The next analysis we perform concerns the energy consumption of each method. We estimate the energy consumption of each method using the running time of the corresponding algorithm for each dataset and the value of K. We assume that each method uses the maximum power of the corresponding computational resources during its running time. Therefore, we derive the power consumption of each method, using the running time and maximum power consumption of CPU and/or GPU, which can be found within

the specification of the corresponding CPU or GPU model. As shown in Figure C.4 in Appendix C.1.2, the Laplacian score feature selection needs the least amount of energy among the methods on all datasets except the MNIST dataset. QuickSelection$_{10}$ is the best performer on MNIST in terms of energy consumption. Laplacian score and MCFS are sensitive to the number of samples. They cannot perform well on MNIST, either in terms of accuracy or efficiency. The maximum memory usage during feature selection for Laplacian score and MCFS on MNIST is 56 GB and 85 GB, respectively. Therefore, they are not a good choice in case of having a large number of samples. QuickSelection is the second-best performer in terms of energy consumption, and also the best performer among the autoencoder-based methods. QuickSelection is not sensitive to the number of samples or the number of dimensions.

Efficiency vs. Accuracy. In order to study the trade-off between accuracy and resource efficiency, we perform another in-depth analysis. In this analysis, we plot the trade-off between accuracy (including, classification and clustering accuracy) and resource requirement (including, memory and energy consumption). The results are shown in Figures 4.6 and 4.7 that correspond to the energy-accuracy and memory-accuracy trade-off, respectively. Each point in these plots refers to the results of a particular combination between a specific method and dataset when selecting 50 features (except Madelon, for which we select 20 features). As can be observed in these plots, QuickSelection, MCFS, and Lap_score usually have a good trade-off between the considered metrics. A good trade-off between a pair of metrics is to maximize the accuracy (classification or clustering accuracy) while minimizing the computational cost (power consumption or memory requirement). However, when the number of samples increases (on the MNIST dataset), both MCFS and Lap_score fail to maintain a low computational cost and high accuracy. Therefore, when the dataset size increases, these two methods are not an optimal choice. Among the autoencoder-based methods, in most cases, QuickSelection$_{10}$ and QuickSelection$_{100}$ are among the Pareto optimal points. Another significant advantage of our proposed method is that it gives the ranking of the features as the output. Therefore, unlike the MCFS or CAE that need the value of K as their input, QuickSelection is not dependent on K and needs just a single training of the sparse DAE model for any values of K. Therefore, the computational cost of QuickSelection is the same for all values of K, and only a single run of this algorithm is required to get the hierarchical importance of features.

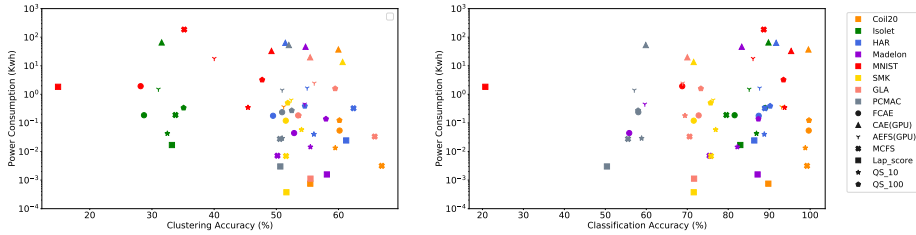


Figure 4.6: Estimated power consumption (Kwh) vs. accuracy (%) when selecting 50 features (except Madelon for which we select 20 features). Each point refers to the result of a single dataset (specified by colors) and method (specified by markers) where the x and y-axis show the accuracy and the estimated power consumption, respectively.

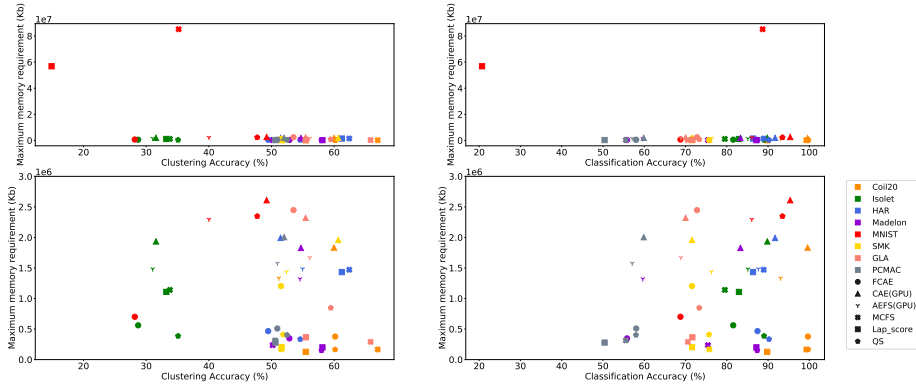


Figure 4.7: Maximum memory requirement (Kb) vs. accuracy (%) when selecting 50 features (except Madelon for which we select 20 features). Each point refers to the result of a single dataset (specified by colors) and method (specified by markers) where the x and y-axis show the accuracy and the maximum memory requirement, respectively. Due to the high memory requirement of MCFS and Lap_score on the MNIST dataset which makes it difficult to compare the other results (upper plots), we zoom in on this section in the bottom plots.

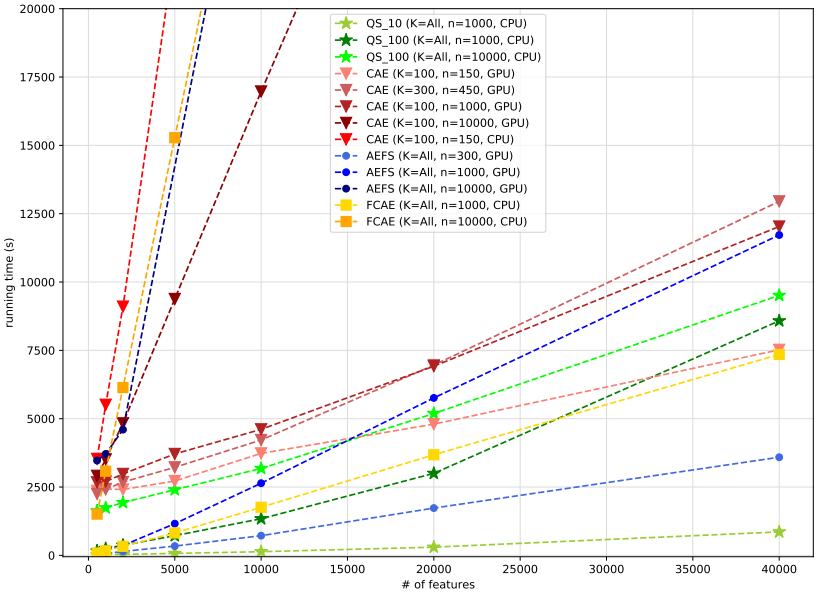


Figure 4.8: Running time comparison on an artificially generated dataset. The features are generated using a standard normal distribution and the number of samples for each case is 5000.

4.5.2 Running Time Comparison on an Artificially Generated Dataset

In this section, we perform a comparison of the running time of the autoencoder-based feature selection methods on an artificially generated dataset. Since on the benchmark datasets both the number of features and samples are different, it is not easily possible to compare clearly the efficiency of the methods. This experiment aims to compare the models' real wall-clock training time in a controlled environment with respect to the number of input features and hidden neurons. In addition, in Appendix C.5, we have conducted another experiment regarding the evaluation of the methods on a very large artificial dataset, in terms of both computational resources and accuracy.

In this experiment, we aim to compare the speed of QuickSelection versus other autoencoder-based feature selection methods for different numbers of input features. We run all of them on an artificially generated dataset with various

numbers of features and 5000 samples, for 100 training epochs (10 epochs for QuickSelection$_{10}$). The features of this dataset are generated using a standard normal distribution. In addition, we aim to compare the running time of different structures for these algorithms. The specifications of the network structure for each method, the computational resources used for feature selection, and the corresponding results can be seen in Figure 4.8.

For CAE, we consider two different values of K. The structure of CAE depends on this value. CAE has two hidden layers including a concrete selector and a decoder that have K and $1.5K$ neurons, respectively. Therefore, by increasing the number of selected features, the running time of the model will also increase. In addition, we consider the cases of CAE with 1000 and 10000 hidden neurons in the decoder layer (manually changed in the code) to be able to compare it with the other models. We also measure the running time of performing feature selection with CAE using only a single CPU core. It can be seen from Figure 4.8 that its execution time is significantly long. The general structures of AEFS, QuickSelection, and FCAE are similar in terms of the number of hidden layers. They are basic autoencoders with a single hidden layer. For AEFS, we considered three structures with different numbers of hidden neurons, including 300, 1000, and 10000. Finally, for QuickSelection and FCAE, we consider two different values for the number of hidden neurons, including 1000 and 10000.

It can be observed that the running time of AEFS with 1000 and 10000 hidden neurons using a GPU, is much larger than the running time of QuickSelection$_{100}$ with similar numbers of hidden neurons using only a single CPU core, respectively. The same pattern is also visible in the case of CAE with 1000 and 10000 hidden neurons. This pattern also repeats in the case of FCAE with 10000 hidden neurons. The running time of FCAE with 1000 hidden neurons is approximately similar to QuickSelection$_{100}$. However, the difference between these two methods is more significant when we increase the number of hidden neurons to 10000. This is mainly due to the fact that the difference between the number of parameters of QuickSelection and the other methods becomes much higher for large values of K. Besides, these observations depict that the running time of QuickSelection does not change significantly by increasing the number of hidden neurons.

As we have also mentioned before, QuickSelection gives the ranking of the features as the output. Therefore, unlike CAE which should be run separately for different values of K, QuickSelection is not affected by the choice of K because it computes the importance of all features at the same time and after finishing the training. In short, QuickSelection$_{10}$ has the least running time among other autoencoder-based methods while being independent of the value of K. In addition, unlike the other methods, the running time of QuickSelection is not

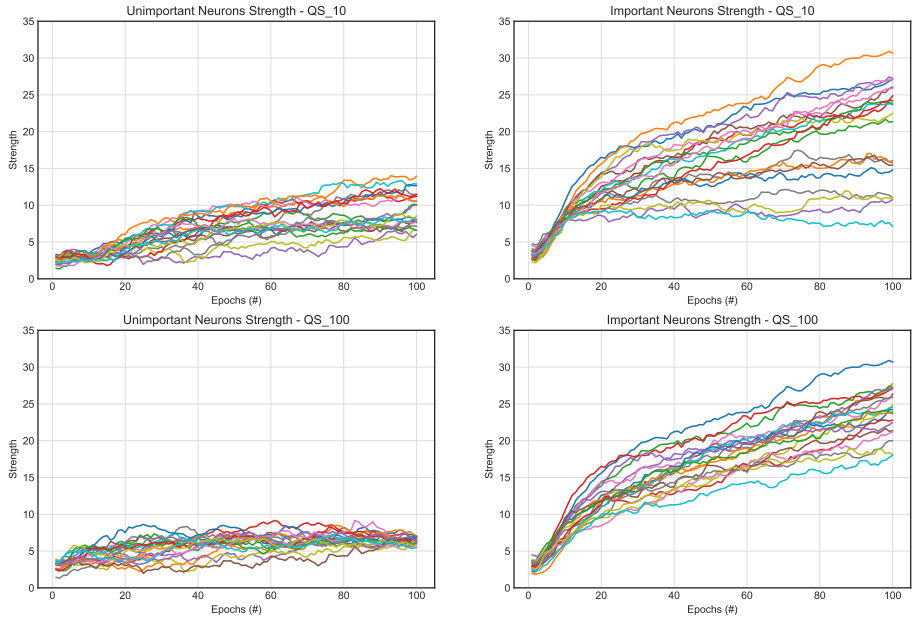


Figure 4.9: Strength of the 20 most informative and non-informative features of the Madelon dataset, selected by QS_{10} and QS_{100}. Each line in the plots corresponds to the strength values of a selected feature by QS_{10}/QS_{100} during training. The features selected by QS_{10} have been observed until epoch 100 to compare the quality of these features with QS_{100}.

sensitive to the number of hidden neurons since the number of parameters is low even for a very large hidden layer.

4.5.3 Neuron Strength Analysis

In this section, we discuss the validity of neuron strength as a measure of the feature importance. We observe the evolution of the network during training to analyze how the neuron strength of important and unimportant neurons changes during training.

We argue that the most important features that lead to the highest accuracy of feature selection are the features corresponding to neurons with the highest strength. In a neural network, weight magnitude is a metric that shows the

importance of each connection [KM98]. This stems from the fact that weights with a small magnitude have a small effect on the performance of the model. At the beginning of training, we initialize all connections to a small random value. Therefore, all the neurons have almost the same strength/importance. As the training proceeds, some connections grow to a larger value while others are pruned from the network during the dynamic connections removal and regrowth of the SET training procedure. The growth of the stable connection weights demonstrates their significance in the performance of the network. As a result, the neurons connected to these important weights contain important information. In contrast, the magnitude of the weights connected to unimportant neurons gradually decreases until they are removed from the network. In short, important neurons receive connections with a larger magnitude. As a result, neuron strength, which is the summation of the magnitude of weights connected to a neuron, can be a measure of the importance of an input neuron and its corresponding feature.

To support our claim, we observe the evolution of neurons' strength on the Madelon dataset. This choice is made due to the distinction between informative and non-informative features in the Madelon dataset. As described earlier, this dataset has 20 informative features, and the rest of the features are non-informative noise. We consider the 20 most informative and non-informative features detected by QS_{10} and QS_{100}, and monitor their strength during training (as observed in Figure 4.3, the maximum accuracy is achieved using the 20 most informative features, while the least accuracy is achieved using the least important features). The features selected by QS_{10} are also being monitored after the algorithm is finished (epoch 10) until epoch 100, in order to compare the quality of the selected features by QS_{10} with QS_{100}. In other words, we extract the index of important features using QS_{10}, continue the training without making any changes in the network, and monitor how the strength of the neurons corresponding to the selected index would evolve after epoch 10. The results are presented in Figure 4.9. At the initialization (epoch 0), the strength of all these neurons is almost similar and below 5. As the training starts, the strength of significant neurons increases, while the strength of unimportant neurons does not change significantly. As can be seen in Figure 4.9, some of the important features selected by QS_{10} are not among those of QS_{100}; this can explain the difference in the performance of these two methods in Table 4.2 and 4.3. However, QS_{10} is able to detect a large majority of the features found by QS_{100}; these features are among the most important ones among the final 20 selected features. Therefore, we can conclude that most of the important features are detectable by QuickSelection, even at the first few epochs of the

algorithm.

4.6 Conclusion

In this chapter, we propose QuickSelection, an energy-efficient unsupervised feature selection algorithm. QuickSelection derives the importance ranking of all input features simultaneously at one round of training. We demonstrate the ability of QuickSelection to (1) achieve the best trade-off between clustering accuracy, classification accuracy, maximum memory requirement, and running time, among other methods considered, (2) significantly reduce memory and computational costs compared to the dense model (3) be the most energy-efficient among the autoencoder-based methods considered (4) perform decently in noisy environments. Therefore, QuickSelection can be a good choice for reducing the costs raised by the high-dimensional and noisy data in resource-constrained environments while being able to find qualitative features. Additionally, this can pave the way for reducing the ever-increasing computational costs of deep learning models imposed on data centers. As a result, this will not only save the energy costs of processing high-dimensional data but also will ease the challenges of high energy consumption imposed on the environment.

Chapter 5

Supervised Feature Selection with Neuron Evolution in Sparse Neural Networks

Despite the promising results of QuickSelection, it performs feature selection post-training in an unsupervised manner, posing challenges in tailoring feature selection for specific classification goals. Additionally, the strength metric may not accurately rank features in a very high-dimensional feature space. To tackle these limitations, in this chapter, we propose a novel resource-efficient supervised feature selection method using sparse neural networks, named NeuroFS, drawing inspiration from evolutionary processes. By gradually pruning the uninformative features from the input layer of a sparse neural network trained from scratch, NeuroFS reduces the search space of features. This reduction in the feature space empowers the strength metric to perform more effectively. Furthermore, by conducting supervised training and integrating neuron updates during the training process, NeuroFS tailors the selected features to the specific classification task at hand. By performing several experiments on 11 *low and high-dimensional real-world benchmarks of different types, we demonstrate that NeuroFS achieves the highest ranking-based*

This Chapter is integrally based on: Zahra Atashgahi, Xuhao Zhang, Neil Kichler, Shiwei Liu, Lu Yin, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu, *Supervised Feature Selection with Neuron Evolution in Sparse Neural Networks,* **Transactions on Machine Learning Research (TMLR)**, ISSN 2835-8856, 2023, URL: *https://openreview.net/forum?id=GcO6ugrLKp*.

score among the considered state-of-the-art supervised feature selection models. The code is available at https://github.com/zahraatashgahi/NeuroFS.

5.1 Introduction

Feature selection has been gaining increasing importance due to the growing amount of big data. The high dimensionality of data can give rise to issues such as the curse of dimensionality, over-fitting, and high memory and computation demands [LCW+18]. By removing the irrelevant and redundant attributes in a dataset, feature selection combats these issues while increasing data interpretability and potentially improving the accuracy [CS14].

The literature on feature selection can be stratified into three major categories: filter, wrapper, and embedded methods [CS14]. Unlike filter methods that perform feature selection before the learning task and wrapper methods that use a learning algorithm to evaluate a subset of the features, embedded methods use learning algorithms to determine the informative features [ZNLW19]. Since embedded methods combine feature selection and the learning task into a unified problem, they usually perform more effectively than the other two categories in terms of the quality of the selected features [HWZ+18, BAZ19]. Therefore, this work focuses on embedded feature selection due to its superior performance.

In recent years, there has been a growing interest in using artificial neural networks (ANNs) to perform embedded feature selection. This is due to their favorable characteristic of automatically exploring non-linear dependencies among input features, which is often neglected in traditional embedded feature selection methods [Tib96]. In addition, the performance of ANNs scales with the dataset size [HNA+17], while most feature selection methods do not scale well on large datasets [LCW+18]. Moreover, many works have demonstrated the efficacy of neural network-based feature selection in both supervised [LFLN18, LRAT21, YLNK20, SY20, WC20] and unsupervised [HWZ+18, CS15, BAZ19, DS19, ASvdL+22, SLSK22] settings.

However, while being effective in terms of the quality of the selected features, feature selection with ANNs is still a challenging task. Over-parameterization of neural networks results in high computational and memory costs, which make their deployment and training on low-resource devices infeasible [HABN+21]. Only very few works have tried to increase the scalability of feature selection using neural networks on low-resource devices. E.g., [ASvdL+22] proposes, for the first time, that sparse neural networks [HABN+21] can be exploited to perform efficient feature selection. Their proposed method, QuickSelection,

which is designed for unsupervised feature selection, trains a sparse neural network from scratch to derive the ranking of the features using the information of the corresponding neurons in the neural network.

In this chapter, by introducing dynamic input neuron evolution into the training of a sparse neural network, we propose to use the sparse neural networks to perform supervised feature selection and introduce an efficient feature selection method, named **F**eature **S**election with **Neuro**n Evolution (**NeuroFS**). Our contributions can be summarized as follows:

- We introduce dynamic neuron pruning and regrowing in the input layer of sparse neural networks during training.
- Based on the newly introduced dynamic neuron updating process, we propose a novel efficient supervised feature selection algorithm named "NeuroFS".
- We evaluate NeuroFS on 11 real-world benchmarks for feature selection and demonstrate that NeuroFS achieves the highest average ranking among the considered feature selection methods on low and high-dimensional datasets.

5.2 Background

In this section, we provide background information on feature selection and sparse neural networks.

5.2.1 Feature Selection

Problem Formulation

In this section, we first describe the general supervised feature selection problem. Consider a dataset $\mathbb{X}$ containing m samples $(\boldsymbol{x}^{(i)}, y^{(i)})$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is the i-th sample in data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times d}$, d is the dimensionality of the dataset or the number of the features, and $y^{(i)}$ is the corresponding label for supervised learning. Feature selection aims to select a subset of the most discriminative and informative features of $\boldsymbol{X}$ as $\mathbb{F}_s \subset \mathbb{F}$ such that $|\mathbb{F}_s| = K$, where $\mathbb{F}$ is the original feature set, and K is a hyperparameter of the algorithm which indicates the number of features to be selected.

Objective function. In supervised feature selection, we seek to optimize:

$$\mathbb{F}_s^* = \underset{\mathbb{F}_s \subset \mathbb{F}, |\mathbb{F}_s| = K}{\arg\min} \sum_{i=0}^{m} J(f(\boldsymbol{x}_{\mathbb{F}_s}^{(i)}; \boldsymbol{\theta}), y^{(i)}), \tag{5.1}$$

where $\mathbb{F}_s^*$ is the final selected feature set, J is a desired loss function, and $f(\boldsymbol{x}_{\mathbb{F}_s}^{(i)}; \boldsymbol{\theta})$ is a classification function parameterized by $\boldsymbol{\theta}$ aiming at estimating the target for the i-th sample using a subset of features $\boldsymbol{x}_{\mathbb{F}_s}^{(i)}$.

Solving this optimization problem can be a challenging task. As the choice of feature subset $\mathbb{F}_s$ grows exponentially when increasing the number of features d, solving Equation 5.1 is an NP-hard problem. Additionally, it is important that function f can learn a fruitful representation and complex data dependencies [LRAT21]. We choose artificial neural networks due to their high expressive power; a simple one-hidden layer feed-forward neural network is known to be a universal approximator [GBC16]. Finally, as we aim to select features in a computationally efficient manner, in this work, we choose sparse neural networks for training and performing feature selection.

Related Work

Feature selection methods are classified into three main categories: filter, wrapper, and embedded methods. **Filter methods** use criteria such as correlation [GE03], mutual information [CS14], Laplacian score [HCN06], to rank the features independently from the learning task, which makes them fast and efficient. However, they are prone to selecting redundant features [CS14]. **Wrapper methods** find a subset of features that maximize an objective function [ZNLW19] using various search strategies such as tree structures [KJ97] and evolutionary algorithms [LS+96]. However, these methods are costly in terms of computation. **Embedded methods** aim to address the drawbacks of the filter and wrapper approaches by integrating feature selection and training tasks to optimize the subset of features. Various approaches have been used to perform embedded feature selection including, mutual information [Bat94, PLD05], the SVM classifier [GWBV02], and neural networks [SL97].

Recently, neural network-based feature selection in both supervised [LFLN18, LRAT21, YLNK20, SY20, WC20] and unsupervised [ASvdL+22, BAZ19, HWZ+18, CS15, DS19] settings have gained increasing attention due to their favorable advantages of capturing non-linear dependencies and showing good performance on large datasets. However, most of these methods suffer from over-parameterization, which leads to high computational costs, particularly on

high-dimensional datasets. QuickSelection [ASvdL+22] addresses this issue by exploiting sparse neural networks. However, due to the random growth of connections in its topology update stage, it might not be able to detect fastly enough the informative features on high-dimensional datasets due to the large search space. As we show in the following sections, we address this issue by gradually pruning uninformative input neurons and exploiting gradients to speed up the learning process.

5.2.2 Sparse Neural Networks

Sparse neural networks have been proposed to address the high computational costs of dense neural networks [HABN+21]. They aim to reduce the parameters of a dense neural network while preserving a decent level of performance on the task of interest.

There are two main approaches to obtain a sparse neural network: dense-to-sparse and sparse-to-sparse methods [MMP+21].

- **Dense-to-sparse** algorithms start with a dense network and prune the unimportant connections to obtain a sparse network [LDS90,HS93,HPTD15, LAT19, FC19, MTK+17, MMT+19, GEH19]. As they start with a dense network, they need the memory and computational resources to fit and train the dense network for at least a couple of iterations. Therefore, they are mostly efficient during the inference phase.

- **Sparse-to-sparse** algorithms aim to bring computational efficiency both during the training and inference. These methods use a static [MMN+16] or dynamic [MMS+18, BKML18] sparsity pattern during training. In the following, we will elaborate on sparse training with dynamic sparsity (or started to be known in the literature as dynamic sparse training (DST)), which usually outperforms the static approach.

Dynamic Sparse Training (DST)

DST is a class of methods to train sparse neural networks sparsely from scratch. DST methods aim at optimizing the sparse connectivity of a sparse neural network during training, such that they never use dense network matrices during training [MMP+21]. Formally, DST methods start with a sparse neural network $f(\boldsymbol{x}, \boldsymbol{\theta}_s)$ with a sparsity level of S. We have $S = 1 - \frac{\|\boldsymbol{\theta}_s\|_0}{\|\boldsymbol{\theta}\|_0}$, where $\boldsymbol{\theta}_s$ is a subset of parameters of the equivalent dense network parameterized by $\boldsymbol{\theta}$, $\|\boldsymbol{\theta}_s\|_0$ and $\|\boldsymbol{\theta}\|_0$ are the

number of parameters of the sparse and dense network, respectively. They aim to optimize the following problem:

$$\boldsymbol{\theta}_s^* = \underset{\boldsymbol{\theta}_s \in \mathbb{R}^{\|\boldsymbol{\theta}\|_0},\ \|\boldsymbol{\theta}_s\|_0 = D\|\boldsymbol{\theta}\|_0}{\arg\min} \frac{1}{m} \sum_{i=1}^{m} J(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_s), \boldsymbol{y}^{(i)}), \tag{5.2}$$

where $D = 1 - S$ is called density level. During training, DST methods periodically update the sparse connectivity of the network; e.g., in [MMS+18, EGM+20] authors remove a fraction ζ of the parameters $\boldsymbol{\theta}_s$ and add the same number of parameters to the network to keep the sparsity level fixed. In the literature, usually, weight magnitude has been used as a criterion for dropping the connections. However, there exists various approaches for weight regrowth including, random [MMS+18, MW19], gradient-based [EGM+20, DYJ19, DZ19, JPR+20], locality-based [HABN+21], and similarity-based [APL+22]. It has been shown that in many cases, they can match or even outperform their dense counterparts [FC19, MMS+18, LCC+21, LYMP21]. [EIKD22] have discussed in-depth that DST methods improve the gradient flow in the network by updating the sparse connectivity that eventually leads to a good performance. In this chapter, we exploit sparse neural network training from scratch to design an efficient supervised feature selection method.

5.3 Proposed Method

In this section, we present our proposed methodology for feature selection using sparse neural networks, named **F**eature **S**election with **Neuro**n evolution (**NeuroFS**). We start by describing our proposed sparse training algorithm. Then, we explain how the introduced sparse training algorithm can be used to perform feature selection.

5.3.1 Dynamic Neuron Evolution

Inspired by the weights update policy in DST, we introduce dynamic neuron evolution in the framework of DST to perform efficient feature selection. While existing DST methods update only the connections or the hidden neurons [DYJ19] to evolve the topology of sparse neural networks, we propose to update also the input neurons of the network to dynamically derive a set of relevant features of the given input data.

Our proposed neuron evolution process has two steps. Consider a network in which only a fraction of input neurons have non-zero connections. We periodically update the input layer connectivity by first dropping a fraction of unimportant neurons (*neuron removal*) and then adding a number of unconnected neurons back to the network (*neuron regrowth*):

- **Neuron Removal.** The criterion used for dropping the neurons is strength, which is introduced in [ASvdL+22]. Strength is the summation of the absolute weights of existing connections for an input neuron. A higher strength of a neuron indicates that the corresponding input feature has higher importance in the data. Therefore, we drop a fraction of low-strength neurons at each epoch. We call the neurons with at least one non-zero weight connection, *active*, and the neurons without any non-zero connections, *inactive*.

- **Neuron Regrowth.** After removing unimportant neurons, we explore the inactive neurons. We activate a number of neurons with the highest potential to enhance the learned data representation. We exploit the gradient magnitude of the non-existing connections for each neuron as a criterion to choose the most important inactive neurons. It has been shown in [EGM+20] that adding the zero-connections with the largest gradients magnitude in the DST process accelerates the learning and improves the accuracy. [EIKD22] also have shown that picking the connections with the highest gradient magnitude increases the gradient flow, which eventually leads to a decent performance. We hypothesize that adding inactive neurons connected to the zero-connections with the highest gradient magnitude to the network would improve the data representation and increase the likelihood of finding an informative set of features.

Dynamic neuron evolution is loosely inspired by evolutionary algorithms [SM02]. Still, due to the large search space, the latter cannot be directly applied to our problem without significantly increased computational time. To alleviate this, we seek inspiration in the dynamics of the evolution process from the biological brain at the epigenetic level, which performs cellular changes (seconds to days time scale) [KBCD14], and not at the phylogenic level (generations time scale) as it is usually performed in evolutionary computing. Accordingly, NeuroFS removes and regrows neurons in the input layer of a sparsely trained neural network based on chosen criteria at each epoch until a reduced optimal set of input neurons remains active in the network. In the next section, we

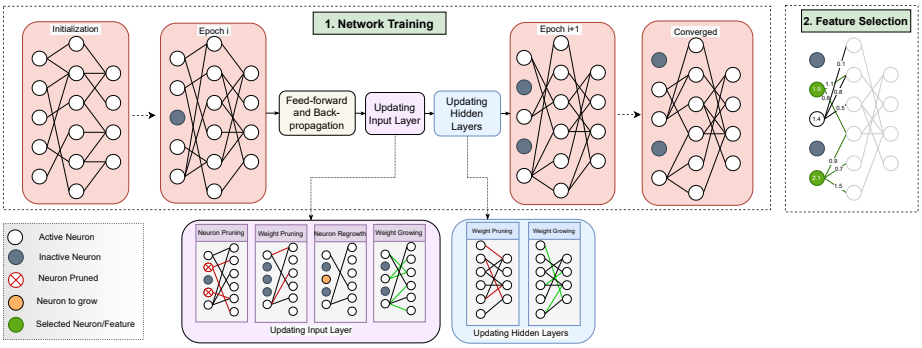


Figure 5.1: Overview of "NeuroFS". At initialization, a sparse MLP is initialized (Section 5.3.2). During training, at each training epoch, after standard feed-forward and back-propagation, input and hidden layers are updated such that a large fraction of unimportant input neurons are gradually dropped while giving a chance to the removed input neurons for regrowth (Section 5.3.2). After convergence, NeuroFS selects the corresponding features to K active neurons with the highest strength (Section 5.3.2).

will explain how NeuroFS uses dynamic neuron evolution to perform feature selection.

5.3.2 NeuroFS

Our proposed algorithm is briefly sketched in Figure 5.1. In short, NeuroFS aims at efficiently selecting a subset of features that can learn an effective representation of the input data in a sparse neural network. In the following, we describe the algorithm in more detail.

Problem Setup

We first start by describing the network structure and problem setup.

Network Architecture. We exploit a supervised deep neural network, Multi-Layer Perceptron (MLP). We initialize a sparse MLP $f(\boldsymbol{x}, \boldsymbol{\theta}_s)$, with L layers and a sparsity level of S.

Initialization. The sparse connectivity is initialized randomly as an Erdos-Renyi random graph [MMS+18]. Sparsity level S is determined by a hyperparameter of the model, named ε, such that the density of layer l is $\varepsilon(n^{(l-1)}+n^{(l)})/(n^{(l-1)}\times n^{(l)})$, and the total number of parameters is equal to $\|\boldsymbol{\theta}_s\|_0=\sum_{l=1}^{L}\left\|\boldsymbol{\theta}_s^{(l)}\right\|_0$, where $l\in\{1,2,...,L\}$ is the layer index and $n^{(l)}$ is number of neurons at layer l. The number of connections at each layer is computed as $\left\|\boldsymbol{\theta}_s^{(l)}\right\|_0=\varepsilon(n^{(l-1)}+n^{(l)})$.

Training

After initializing the network, we start the training process. In summary, we start with a sparse neural network and aim to optimize the topology of the network and the selected subset of features simultaneously. During training, we gradually remove the input neurons while giving a chance for the inactive neurons to be re-added to the network. Finally, when the training is finished, we select the important features from a limited number of active neurons. In the following, we describe the training algorithm in more detail.

At each training epoch, NeuroFS performs the following three steps:

Step 1. Feed-forward and Back-propagation. At each epoch, first, standard feed-forward and back-propagation are performed to train the weights of the sparse neural network.

Step 2. Updating Input Layer. After each training epoch, we update the input layer. The novelty of our proposed algorithm lies mainly in updating the input layer. During training, NeuroFS gradually decreases the number of active input features. In short, at epoch t, it gradually prunes a number of input neurons ($c_{prune}^{(t)}$) and regrows a number of unconnected neurons ($c_{grow}^{(t)}$) back to the network. Updating the input layer in NeuroFS consists of two phases:

- **Removal Phase.** From the beginning of the training until $t_{removal}$, updating the input layer is at the removal phase. In this phase, the total number of active neurons decreases at each epoch such that $c_{prune}^{(t)} > c_{grow}^{(t)}$, if $t \leqslant t_{removal}$. We have $t_{removal} = \lceil \alpha t_{max} \rceil$, where $0 < \alpha < 1$ is a hyperparameter of NeuroFS determining the neuron removal phase duration, $\lceil\rceil$ is the ceiling function, and t_{max} is the total number of epochs.

- **Update Phase.** From $t_{removal}$ until the end of the training, the number of connected neurons remains fixed in the network and only a fraction of neurons are updated. In other words, $c_{prune}^{(t)} = c_{grow}^{(t)}$, if $t > t_{removal}$.

Formally, we compute $c_{prune}^{(t)}$ at epoch (t) as follows:

$$c_{prune}^{(t)} = \begin{cases} c_{remove}^{(t)} + c_{grow}^{(t)}, & t \leqslant t_{removal} \\ c_{grow}^{(t)}, & \text{otherwise} \end{cases}. \tag{5.3}$$

$c_{prune}^{(t)}$ in the removal phase consists of two parts: $c_{remove}^{(t)}$ and $c_{grow}^{(t)}$. As the overall number of active neurons is decreasing in this phase, $c_{remove}^{(t)}$ extra neurons to the updated ones will be removed at each epoch. $c_{remove}^{(t)}$ is computed as:

$$c_{remove}^{(t)} = \lceil \frac{R - R^{(t)}}{t_{removal} - t} \rceil, \tag{5.4}$$

$$R^{(t)} = \sum_{i=1}^{t-1} c_{remove}^{(i)}, \tag{5.5}$$

$$R = \lceil (1 - \zeta_{in}) d - K \rceil, \tag{5.6}$$

where $R^{(t)}$ is the total number of inactive neurons at epoch t, R is the total number of neurons to be removed in the removal phase, and $\zeta_{in} \in \mathbb{R}, 0 < \zeta_{in} < 1$ is the update fraction of the input layer. In other words, the total number of active neurons after the removal phase is $\zeta_{in} d + K$. We keep $\zeta_{in} d$ neurons extra to the number of selected features K, so that the update phase does not disturb the already found important features.

Finally, the number of neurons to grow at epoch t is computed as:

$$c_{grow}^{(t)} = \lceil \zeta_{in} (1 - \frac{t}{t_{max}}) R^{(t)} \rceil. \tag{5.7}$$

In other words, at each epoch, we add a fraction ζ_{in} of the inactive neurons back to the network. However, as the number of inactive neurons increases during training, the number of updated neurons will increase consequently. A large number of updated neurons might diverge the network training. Therefore, we decrease the update fraction linearly during training. At epoch t, we update $\zeta_{in}(1 - \frac{t}{t_{max}})$ proportion of the total inactive neurons.

After computing $c_{prune}^{(t)}$ and $c_{grow}^{(t)}$, the input layer is updated as follows:

1. **Neuron pruning:** $c_{prune}^{(t)}$ neurons with lowest strength are dropped from the input layer. The strength of input neuron i is computed as $s_i = \left\| \boldsymbol{w}^{(i)} \right\|_1$, where $\boldsymbol{w}^{(i)}$ is the weights vector of neuron i.

2. **Weight pruning:** a fraction ζ_{in} of connections with the lowest magnitudes are dropped from the active input features.

3. **Neuron regrowth:** $c_{grow}^{(t)}$ neurons are selected for being activated and added to the network. As discussed in Section 5.3.1, these neurons are the ones connected to the connections with the largest absolute gradient among all non-existing connections of inactive neurons.

4. **Weight growing:** The same number as the number of removed connections will be added to the network so that the sparsity remains fixed during training. These connections are the ones with the largest absolute gradient among all non-existing connections of the active neurons at the current epoch.

Step 3. Updating Hidden Layers. Hidden layers will be updated by updating the sparse connectivity, which is the standard approach in the DST process. We use gradients for weight regrowth [EGM+20]. For each hidden layer $h^{(l)}$, NeuroFS performs the following two steps:

1. **Weight pruning:** a fraction ζ_h of connections with the lowest magnitude are dropped from layer $h^{(l)}$.

2. **Weight growing:** the same number as the number of removed connections will be added to layer $h^{(l)}$. These connections are the ones with the largest absolute gradient among all non-existing connections.

Feature Selection

After the training process is finished, we perform feature selection. We select K neurons with the highest strength out of the $\zeta_{in}d + K$ remaining active neurons. The corresponding features to these K neurons are the most informative and relevant features in our dataset. NeuroFS is schematically described in Figure 5.1 and the corresponding pseudo-code is available at Algorithm 5.

Algorithm 5 NeuroFS

Input: Dataset $\mathbb{X}$, sparsity hyperparameter ε, drop fractions ζ_{in} and ζ_h, neuron removal phase duration hyperparameter α, number of training epochs t_{max}, number of features to select K.
Initialization: Initialize the network with sparsity level S determined by ε (Section 5.3.2)
for $t \in \{1, \ldots, \#t_{max}\}$ **do**
    I. Standard feed-forward and back-propagation
    II. Update Input Layer:
    0. Compute $c^{(t)}_{prune}$ (Equation 5.3) and $c^{(t)}_{grow}$ (Equation 5.7).
    1. Drop $c^{(t)}_{prune}$ neurons with the lowest strength.
    2. Drop a fraction ζ_{in} of connections with the lowest magnitude.
    3. Select $c^{(t)}_{grow}$ inactive neurons (that have connections with the highest gradient magnitude), to be activated.
    4. Regrow as many connections as have been removed to the active neurons.
    III. Update Hidden Layers:
    for $l \in \{1, \ldots, L\}$ **do**
        1. Drop a fraction ζ_h of connections with the lowest magnitude from layer h^l.
        2. Regrow as many connections as have been removed in layer h^l.
Feature Selection: Select K features corresponding to the active neurons with the highest strength in the input layer.

5.4 Experiments and Results

In this section, we first describe the experimental settings and then analyze the performance of NeuroFS and compare it with several state-of-the-art feature selection methods.

5.4.1 Settings

This section describes the experimental settings, including, datasets, compared methods, hyperparameters, implementation, and the evaluation metric.

Table 5.1: Datasets characteristics.

Dataset	Type	# Features	# Samples	# Train	# Test	# Classes
COIL-20	Image	1024	1440	1152	288	20
USPS		256	9298	7438	1860	10
MNIST		784	70000	60000	10000	10
Fashion-MNIST		784	70000	60000	10000	10
Isolet	Speech	617	7737	6237	1560	26
HAR	Time Series	561	10299	7352	2947	6
BASEHOCK	Text	4862	1993	1594	399	2
Arcene	Mass Spectrometry	10000	200	160	40	2
Prostate_GE	Biological	5966	102	81	21	2
SMK-CAN-187		19993	187	149	38	2
GLA-BRA-180		49151	180	144	36	4

Datasets

We evaluate the effectiveness of NeuroFS on eleven datasets[1] described in Table 5.1.

Comparison

We have selected seven state-of-the-art feature selection methods for comparison as follows:

Embedded methods: LassoNet [LRAT21] exploits a neural network with residual connections to the input layer and solves a two-component (linear and non-linear) optimization problem to find the feature importance. STG [YLNK20] exploits a continuous relaxation of Bernoulli distribution in a neural network to perform feature selection. QuickSelection [ASvdL+22] (denoted as QS in the Figures) selects features using the strength of input neurons of a sparse neural network. RFS [NHCD10] employs a joint $\ell_{2,1}$-norm minimization on the loss function and regularization to select features.

Filter methods: Fisher_score [GLH11] selects features that maximizes similarity of feature values among the same class. CIFE [LT06] maximizes the conditional redundancy between unselected and selected features given the class labels. Finally, ICAP [Jak05] iteratively selects features maximizing the mutual information with the class labels given the selected features.

[1]Available at *https://jundongl.github.io/scikit-feature/datasets.html*

Hyperparameters

The architecture of the network used in the experiments is a 3-layer sparse MLP with 1000 neurons in each hidden layer. The activation function used for the hidden layers is *Tanh* (except for Isolet dataset where *Relu* is used), and the output layer activation function is *Softmax*. The values for the hyperparameters were found through a grid search among a small set of values. We have used stochastic gradient descent (SGD) with a momentum of 0.9 as the optimizer. The parameters for training neural network-based methods, including batch size, learning rate, and the number of epochs (t_{max}), have been set to 100, 0.01, and 100, respectively. However, the batch size for datasets with few samples ($m \leq 200$) was set to 20. The hyperparameter determining the sparsity level ε is set to 30. Update fraction for the input layer ζ_{in} and hidden layer ζ_h have been set to 0.2 and 0.3 respectively. Neuron removal duration hyperparameter α is set to 0.65. ζ_{in} and α are the only hyperparameters particular to NeuroFS. We use min-max scaling for data preprocessing for all methods except for the BASEHOCK dataset, where we perform standard scaling with zero mean and unit variance.

Implementation

We implemented our proposed method using Keras [C+15]. The starting point of our implementation is based on the sparse evolutionary training introduced as SET in [MMS+18][2] to which we added the gradient-based connections growth proposed in RigL [EGM+20]. For Fisher_score, CIFE, ICAP, and RFS, we have used the implementations provided by the *Scikit-Feature* library [LCW+18][3]. The hyperparameter of RFS (γ) has been set to 10 (searched among $[0.01, 0.1, 0.5, 1, 10]$). We implemented QuickSelection [ASvdL+22] in our code; we adapted it to supervised feature selection, as this was not done in QuickSelection. We have used a similar structure and sparsity level ($\epsilon = 30$) to our method for a fair comparison. For QuickSelection, we set $\zeta = 0.3$. For STG and LassoNet, we used the implementation provided by the authors[4][5]. For STG, we used a 3-layer MLP with 1000 hidden neurons in each layer and set the hyperparameter $\lambda = 0.5$ (searched among $[0.001, 0.01, 0.5, 1, 10]$)). For LassoNet, we used a 1-layer MLP with 1000 hidden neurons and set $M = 10$, as suggested by the authors [LRAT21]. Please

[2] *https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks*
[3] *https://jundongl.github.io/scikit-feature/*
[4] *https://github.com/lasso-net/lassonet*
[5] *https://github.com/runopti/stg*

note that we have also tried using a 3-layer MLP for LassoNet. However, it significantly increased the running time, and particularly on large datasets, it exceeded the 12-hour running time. In addition, in the other cases, it did not lead to significantly different results than LassoNet with 1-layer MLP. To have a fair comparison, for NN-based methods (NeuroFS, LassoNet, STG, and QS), we used similar training hyperparameters, including learning rate (0.01), optimizer (SGD), batch size (100, except 20 for datasets with few samples (m<=200)), and training epoch (100). We consider a 12-hour limit on the running time of each experiment. The results of the experiments that exceed this limit are discarded. We used a *Dell R730* processor to run the experiments. We run neural network-based methods using *Tesla-P100* GPU with 16G memory.

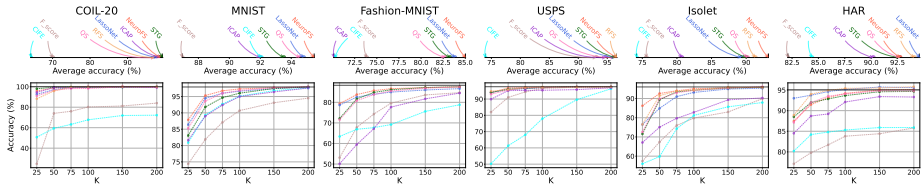
Evaluation Metrics

For evaluating the methods, we use classification accuracy of a *SVM* classifier [KSBM01] with RBF kernel implemented by Scikit-Learn library[6] and used the default hyperparameters of this library. As some of the compared methods do not exploit neural networks to perform feature selection, we intentionally use a non-neural network-based classifier to ensure that the evaluation process is objective and does not take advantage of the same underlying mechanisms as our method. We first find the K important features using each method. Then, we train a SVM classifier on the selected features subset of the training set. We report the classification accuracy on the test set as a measure of performance. We have also evaluated the methods using two other classifiers including KNN and ExtraTrees in Appendix D.3. We have considered classification accuracy using all features as the *baseline* method.

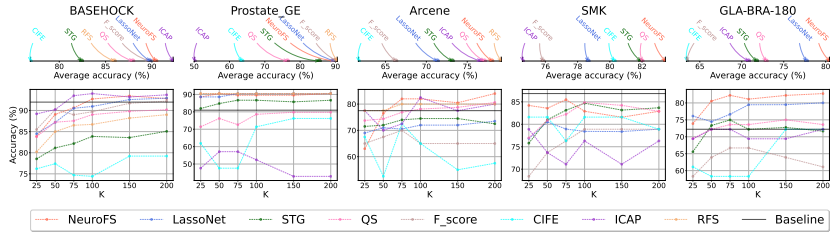
5.4.2 Feature Selection Evaluation

In this section, we evaluate the performance of NeuroFS and compare it with several feature selection algorithms. We run all the methods on the datasets described in Section 5.4.1 and for several values of $K \in \{25, 50, 75, 100, 150, 200\}$. Then, we evaluate the quality of the selected set of features by measuring the classification accuracy on an unseen test set as described in Section 5.4.1. The results are an average of five different seeds. The detailed results for low and high-dimensional datasets, including accuracy for various values of K (below) and average accuracy over K (above), are demonstrated in Figure 5.2. We have

[6] *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*



(a) Low-dimensional datasets



(b) high-dimensional datasets

Figure 5.2: Supervised feature selection comparison for low (a) and high-dimensional (b) datasets, including accuracy for various values of K (below) and average accuracy over K (above).

also presented the detailed results for each value of K in Table D.4 in Appendix D.5. To summarize the results and have a general overview of the performance of each method independent of a particular K value, we have shown the average accuracy over the different values of K in Table 5.2.

As presented in Figure 5.2 and Table 5.2, NeuroFS is the best performer

Table 5.2: Supervised feature selection comparison (average classification accuracy over various K values (%)). Empty entries show that the corresponding experiments exceeded the time limit (12 hours). Bold and italic fonts indicate the best and second-best performer, respectively.

	Low-dimensional Datasets						High-dimensional Datasets				
Method	**COIL-20**	**MNIST**	**F-MNIST**	**USPS**	**Isolet**	**HAR**	**BASEHOCK**	**Prostate_GE**	**Arcene**	**SMK**	**GLA**
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	*98.79±0.22*	**95.48±0.34**	**85.03±0.15**	**96.68±0.16**	**93.22±0.11**	92.74±0.23	*90.42±0.80*	89.70±0.72	*78.00±1.78*	**82.36±0.98**	**80.46±0.99**
LassoNet	98.03±0.31	*94.80±0.23*	*83.81±0.12*	96.41±0.05	89.31±0.13	**94.63±0.10**	89.77±0.56	*89.86±0.78*	71.33±1.94	78.67±2.09	*77.70±1.84*
STG	**99.30±0.31**	94.16±0.47	83.74±0.41	*96.55±0.17*	89.13±1.43	91.87±0.63	85.48±0.78	85.41±2.72	73.75±2.78	81.34±2.68	71.19±2.33
QS	97.23±1.34	94.57±0.35	82.69±0.24	96.22±0.20	90.20±1.23	92.70±0.57	87.93±0.40	76.39±7.44	77.08±1.56	*82.01±2.69*	72.91±0.69
Fisher_score	70.02±0.00	86.95±0.00	73.85±0.00	93.12±0.00	75.58±0.00	82.10±0.00	89.72±0.00	**90.50±0.00**	66.25±0.00	75.85±0.00	63.43±0.00
CIFE	64.18±0.00	92.07±0.00	70.27±0.00	73.90±0.00	74.15±0.00	84.38±0.00	76.85±0.00	63.48±0.00	61.67±0.00	80.27±0.00	63.40±0.00
ICAP	98.67±0.00	92.00±0.00	70.12±0.00	94.75±0.00	80.72±0.00	90.20±0.00	**92.30±0.00**	50.00±0.00	76.67±0.00	74.57±0.00	70.80±0.00
RFS	97.28±0.00	-	-	**96.68±0.00**	*91.32±0.00*	*94.08±0.00*	85.93±0.00	**90.50±0.00**	**79.17±0.00**	-	-

in 6 datasets out of 11 considered datasets in terms of average accuracy, while performing very closely to the best performer in the remaining cases. Filter methods, such as ICAP, CIFE, and F-score, have been outperformed by embedded methods on most datasets considered, as they select features independently from the learning task. Among these methods, ICAP performs well on the text dataset (BASEHOCK); this shows that mutual information is informative in feature selection from the text datasets. Among the considered embedded methods, RFS fails to find the informative features on datasets with a high number of samples (e.g., MNIST, Fashion-MNIST) or dimensions (e.g., SMK, GLA-BRA-180) within the considered time limit.

By looking into the results of all considered methods, it can be observed that neural network-based feature selection methods outperform classical feature selection methods in most cases. Therefore, it can be concluded that the complex non-linear dependencies extracted by the neural network are beneficial for the feature selection task. However, as will be discussed in Section 5.5.2, the over-parameterization in dense neural networks, as used for STG and LassoNet, leads to high computational costs and memory requirements, particularly on high-dimensional datasets. NeuroFS and QuickSelection address this issue by exploiting sparse layers instead of dense ones.

NeuroFS outperforms QuickSelection, which is the sparse competitor of NeuroFS, in terms of average accuracy, particularly on the high-dimensional datasets. This is because, for high-dimensional datasets, QuickSelection needs more training time to find the optimal topology in the large connections search space due to the random search. NeuroFS alleviate this problem by exploiting the gradient of the connections to find the informative paths in the network while removing the uninformative neurons gradually to reduce the search space.

To summarize the results and have a general overview of the methods' performance, we use a ranking-based score. For each dataset and value of K, we rank the methods based on their classification accuracy and give a score of 0 to the worst performer, and the highest score ($\#methods - 1$) to the best performer. For each method, we compute the average score for different values of K and different datasets. The results are summarized in Figure 5.3. NeuroFS achieves

(a) Low-dimensional dataset (b) high-dimensional dataset

Figure 5.3: Average ranking score over all datasets and K values.

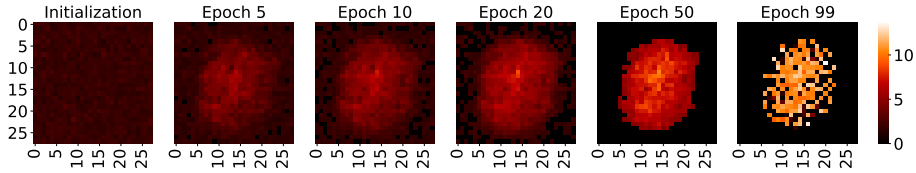


Figure 5.4: Feature importance visualization on the MNIST dataset (number of selected features K=50).

the highest average ranking on both low and high-dimensional datasets.

Overall, it can be concluded that inspired by the evolutionary process, NeuroFS can find an effective subset of features by dynamically changing the sparsity pattern in both input neurons and connections. By dropping the unimportant input neurons (based on magnitude) and adding new neurons based on the incoming gradient, it can mostly outperform its direct competitors, LassoNet, STG, and QuickSelection, in terms of accuracy while being efficient by using sparse layers instead of dense over-parameterized layers.

5.4.3 Feature Importance Visualization

In order to gain a better understanding of the NeuroFS algorithms, in this section, we analyze the feature importance during the training of the network. We run NeuroFS on the MNIST dataset and for $K = 50$ and visualize the strength of input neurons as a heat-map at several epochs in Figure 5.4.

As shown in Figure 5.4, at the initialization, all the neurons have very close strength/importance. This stems from the random initialization of the weights to a small random value. During training, the number of active neurons gradually decreases. The removed neurons are mostly located towards the edges of this picture. This pattern is similar to the MNIST digits dataset, where most digits appear in the middle of the image. Finally, at the last epoch, a limited number of neurons have remained active. We select the most important features out of the active features. In conclusion, this experiment shows that NeuroFS can determine the most important region in the features accurately.

5.5 Discussion

In this section, we present the results of several analyses on the performance of NeuroFS, including robustness evaluation and hyperparameter's effect. We have additionally analyzed weight/neuron growth policy in Appendix D.1, and compared NeuroFS with two HSICLasso-based feature selection methods and RigL in Appendix D.2 and D.4, respectively.

5.5.1 Robustness Evaluation: Topology Variation

In this section, we analyze the robustness of NeuroFS to variation in the topology. We aim to explore if different runs of NeuroFS converge to similar or distant topologies and whether NeuroFS performance remains stable for these different topologies.

To achieve this aim, we conduct two experiments. In the first experiment, we analyze the topology of five networks that are trained and initialized with different random seeds. In other words, they start with different sparse connectivities at initialization and have different training paths. In the second experiment, we analyze the topology of five networks initialized with the same sparse connectivity (using a similar random seed) and trained with different random seeds. For both experiments, we measure the topology distance among networks using a metric introduced in [LvdLY+20], called NNSTD. It measures the distance of two sparse networks; NNSTD of 0 means that two networks are identical, and 1 means completely different.

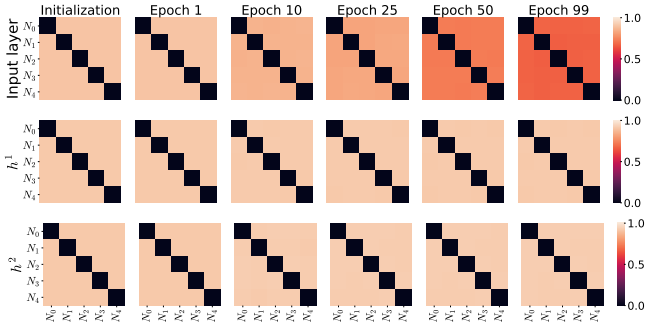
We perform both experiments on the MNIST dataset to find the $K = 50$ most important features. The topology distance of the networks at different epochs are depicted in Figure 5.5 as 2d heatmaps. Each row depicts the distances for one layer of different networks. Each tile in the heatmaps refers to the distance between two layers of two networks. In these figures, N_i refers to the network trained with i^{th} random seed. The corresponding accuracies are shown in Table 5.3.

In Figure 5.5a, the networks are very distant at the beginning as their sparse connectivity (topology) initialized differently. During training, while their hidden layers remain distant, their input layers become more similar. Considering these figures and comparing them with the results in Table 5.3, it can be observed that while the feature selection remains almost the same, the network topologies do not. This indicates that NeuroFS can find several well-performing networks.

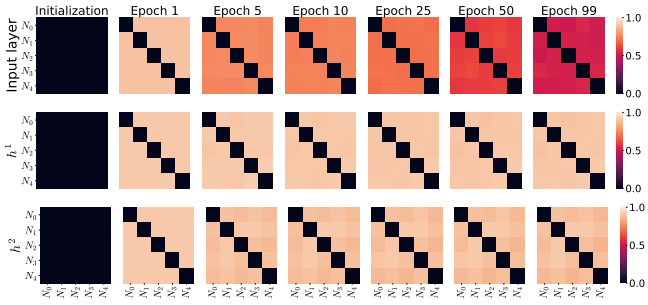
The similarity of the network topologies in Figure 5.5b almost match the pattern of Figure 5.5a. While the networks start from the same sparse connectivity,

they become distant at the next epoch when they start training with different random seeds. This indicates that NeuroFS explores various connectivities during training. Interestingly, in the end, the converged input layers are more similar to each other than the experiment 1, due to the similar sparse connectivity at initialization. As shown in Table 5.3, the corresponding accuracies are close together. Experiment 2 confirms the observations in experiment 1, where NeuroFS finds distant topologies with very close feature selection performance.

To conclude, NeuroFS is robust to changes in topology. While it finds very different topologies overall, the input layers converge to relatively similar topologies, resulting in close feature selection performance.



(a) Different initial sparse connectivity



(b) Similar initial sparse connectivity

Figure 5.5: Topology distance of five MLPs with (a) different and (b) similar initial sparse connectivity (topology). Input layers converge to relatively similar topologies in both cases, while hidden layers remain distant. N_i refers to the network trained with i^{th} random seed.

Table 5.3: NeuroFS Classification Accuracy (%) on the MNIST dataset for five networks ($K = 50$).

	N_0	N_1	N_2	N_3	N_4
NeuroFS (Different initial sparse connectivity)	95.6	95.3	95.5	94.6	95.2
NeuroFS (Similar initial sparse connectivity)	95.6	94.4	96.2	95.4	95.8

5.5.2 Computational Efficiency of NeuroFS

In this section, we analyze the computational efficiency of NeuroFS. We present the number of training FLOPs and the number of parameters of NeuroFS and compare it with its neural network-based competitors.

Estimating the FLOPs (floating-point operations) and parameter count is a commonly used approach to analyze the efficiency gained by a sparse neural network compared to its dense equivalent network [EGM+20, SMM+21]. *Number of parameters* indicates the size of the model, which directly affects the memory consumption and also computational complexity. *FLOPs* estimates the time complexity of an algorithm independently of its implementation. In addition, since existing deep learning hardware is not optimized for sparse matrix computations, most methods for obtaining sparse neural networks only simulate sparsity using a binary mask over the weights. Consequently, the running time of these methods does not reflect their efficiency. Besides, developing proper pure sparse implementations for sparse neural networks is currently a highly researched topic pursued by the community [Hoo21]. Thus, as our work is, in its essence, theoretical, we decided to let this engineering research aspect for future work. Therefore, we also use parameters and FLOPs count to analyze efficiency.

To give an intuitive overview of the efficiency of NeuroFS, we compare NeuroFS with its neural network-based competitors. We compute the FLOPs and number of parameters of two dense MLPs with one ($Dense_1$) and three hidden layers ($Dense_3$). These are the architectures used by LassoNet and STG, respectively. However, it should be noted that LassoNet might require several rounds of training for the dense model. Therefore, we have also computed the actual training FLOPs for LassoNet. In addition, as the computational cost of QuickSelection is similar to our method, we refer to both NeuroFS and QuickSelection as $Sparse$.

As explained in Section 5.3.2, the sparsity/density level is determined by the ε. The density level of the network (D), the number of parameters and FLOP count of NeuroFS, and the compared methods are shown in Table 5.4. We

estimate the FLOP count for the considered methods, using the implementation provided by [EGM+20].

As can be seen from Table 5.4, NeuroFS and QuickSelection ($Sparse$) have the least number of parameters and FLOPs among the considered architectures on all considered datasets, particularly on high-dimensional datasets. In addition, as discussed in Section 5.4.2, NeuroFS outperforms LassoNet, STG, and QuickSelection, in terms of accuracy in most cases considered. In short, NeuroFS is efficient in terms of memory requirements and computational costs while finding the most informative subset of the features on real-world benchmarks, including low and high-dimensional datasets.

5.5.3 Hyperparameters Effect

In this section, we analyze the effect of hyperparameters of NeuroFS on the quality of the selected features. The hyperparameters include neuron removal duration fraction α, hyperparameter determining sparsity level ε, and the update fraction of the input layer ζ_{in}. We try different sets of values for each of these hyperparameters and measure the performance of NeuroFS when selecting $K = 100$ features. The results are presented in Figure 5.6.

The results of most datasets are stable for different sets of hyperparameter values. However, high-dimensional datasets with few samples ($d \geq 10000$ and $m \leq 200$) are sensitive to the sparsity level hyperparameter. The feature selection performance decreases for higher densities; this might come from over-fitting of

Table 5.4: Number of parameters ($\times 10^5$) and Number of training FLOPs ($\times 10^{12}$) of NeuroFS ($Sparse$) and the equivalent dense MLPs on different datasets.

		#parameters ($\times 10^5$)			#FLOPs ($\times 10^{12}$)			
Dataset	$Density$	$Sparse$	$Dense_1$	$Dense_3$	$Sparse$	$Dense_1$	$Dense_3$	LassoNet
COIL-20	6.29%	1.91	10.34	30.34	0.13	0.72	2.10	4.5
MNIST	6.57%	1.84	7.94	27.94	6.66	28.60	100.64	371.0
Fashion-MNIST	6.57%	1.84	7.94	27.94	6.66	28.60	100.64	439.8
USPS	7.40%	1.68	2.66	22.66	0.76	1.19	10.12	10.9
Isolet	7.36%	1.95	6.43	26.43	0.73	2.41	9.90	23.6
HAR	6.73%	1.73	5.67	25.67	0.77	2.50	11.33	20.3
BASEHOCK	4.34%	2.98	48.64	68.64	0.36	5.82	8.21	21.4
Arcene	3.77%	4.52	100.02	120.02	0.05	1.20	1.44	5.8
Prostate_GE	4.15%	3.31	59.68	79.68	0.02	0.37	0.49	1.9
SMK-CAN-187	3.42%	7.52	199.95	219.95	0.07	1.79	1.97	4.4
GLA-BRA-180	3.18%	16.29	491.55	511.55	0.18	5.31	5.52	12.6

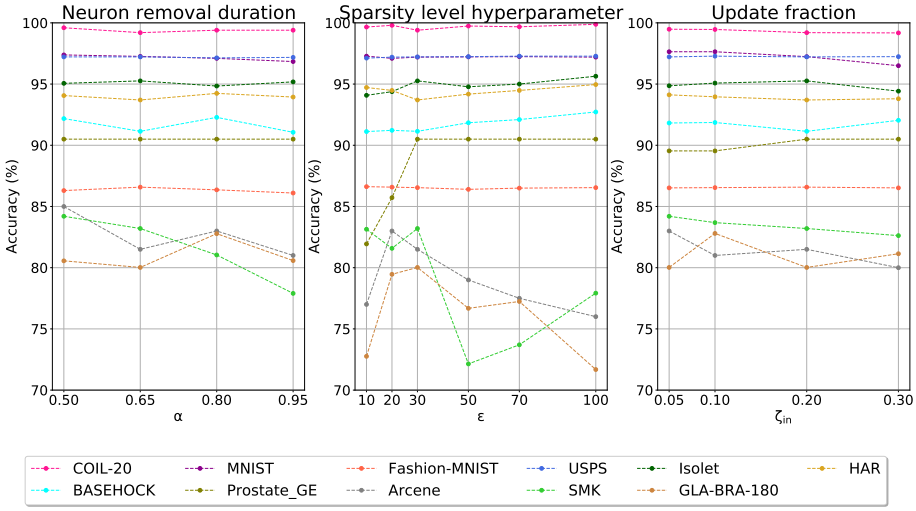


Figure 5.6: Effect of hyperparameters on the performance of the algorithm ($K = 100$).

the network for large parameter counts and a low number of training samples. We select $\alpha = 0.65$, $\varepsilon = 30$, and $\zeta_{in} = 0.2$ as the final values for the other experiments.

5.6 Conclusion

This chapter proposes a novel supervised feature selection method named NeuroFS. NeuroFS introduces dynamic neuron evolution in the training process of a sparse neural network to find an informative set of features. By evaluating NeuroFS on real-world benchmark datasets, we demonstrated that it achieves the highest ranking-based score among the considered state-of-the-art supervised feature selection models. We demonstrated that NeuroFS converges to a similar input layer topology with different runs, showing the feature selection robustness to the network topology. Overall, NeuroFS can be an effective and efficient tool for feature selection in machine learning applications. It is interesting to study in the future what other metrics can be used to measure neuron importance for dynamic weight neuron updating.

Chapter 6

Conclusion, Impact and Future Work

In this chapter, we summarize the main contributions of this thesis. We reflect on the research questions raised in the introduction and provide insights into the limitations, preliminary impacts, future directions, and potential extensions of the work. We emphasize the significance of developing cost-effective artificial neural networks and the potential impact of the proposed methods in advancing the field of deep learning.

6.1 Conclusions

The pursuit for optimal performance in deep neural networks has driven a surge in network size, resulting in increasing computational, memory, and energy demands. This exponential growth, while yielding impressive results, emphasizes the urgent need for efficiency improvements in AI research. High-dimensional data, common in complex real-world problems, exacerbates these challenges, introducing issues such as increased computational complexity, heightened memory consumption, and more intensive energy usage. As outlined in [SGM20], researchers should prioritize computationally efficient algorithms, particularly for deep learning models. Balancing performance and resource constraints remains a critical challenge in the field. To pursue this, in this thesis, we introduced Cost-effective Artificial Neural Networks (CeANNs) as ANNs that can achieve

a targeted performance on the task of interest using minimum computational power and memory and as a result minimum energy consumption (Definition 1.1).

In this thesis, we studied CeANNs from two main perspectives: model and data. In the pursuit of CeANNs, we raised three research questions in Section 1.3. We provided answers to each of these questions in two parts.

6.1.1 Advancing Model Efficiency through Sparsity.

In Chapters 2 and 3, we showed that ANNs can be designed and trained more efficiently, on various data types, compared to over-parameterized dense neural networks that are considered to be computationally expensive.

(Q1) Reducing the costs of training and deploying ANNs. In Chapter 2, we sought the answer to research question 1, which focused on designing ANNs that can be trained and deployed efficiently in terms of computational and memory resources. We exploited the dynamic sparse training (DST) framework to design such networks. In Chapter 2, by taking inspiration from the Hebbian learning theory, we designed a new DST algorithm, called the Cosine Similarity-based and Random Topology Exploration (CTRE). CTRE utilizes the cosine similarity between neuron activations to determine the importance of non-existing connections. By leveraging cosine similarity and random search to evolve the topology in a sparse neural network it addresses several challenges of existing DST algorithms such as slow convergence speed and being computationally expensive (e.g. due to dense gradient computation). Extensive experiments across diverse datasets, including tabular, image, and text data, demonstrate CTRE's superiority over existing sparse training algorithms, particularly in scenarios with high network sparsity. The results also highlight its potential for efficiently training MLPs for tabular data, which is a significant computational workload in data centers. By dynamically evolving network topologies inspired by Hebbian learning, CTRE offers a promising solution for efficiently developing and deploying artificial neural networks across various domains, emphasizing both performance and eco-friendliness.

(Q2) Learning from time series data efficiently. With the ever-increasing size of the collected time series data, cost-effective models for time series analysis are essential to mitigate the rising expenses associated with overly complex deep learning models. In Chapter 3, we explored the challenges and opportunities

in achieving efficient time series forecasting using sparse neural networks, with a specific focus on transformers. While transformers have shown remarkable performance in various time series analysis tasks, their computational demands have made them less feasible for resource-limited applications. We proposed PALS that can decrease the computational and memory costs of training and deploying DNNs for time series forecasting by automatically finding a good trade-off between model size and prediction performance. The experiments conducted on six benchmark datasets and five state-of-the-art transformer variants for time series forecasting demonstrated that PALS significantly reduces model size and computational requirements while maintaining comparable performance to the dense counterparts in terms of prediction loss. The results showed that PALS outperforms other methods to obtain sparse neural networks like GraNet, RigL, and GMP in terms of both loss and sparsity, making it a promising approach for achieving efficient time series forecasting with sparse neural networks. This research opens up possibilities for applying PALS in resource-constrained environments and further advancing the efficiency of time series forecasting with DNNs.

6.1.2 Leveraging Feature Selection for Designing Cost-effective Models.

(Q3) Addressing the challenges of high-dimensional data efficiently. The second part of the thesis, presents solutions to address existing challenges of the high-dimensional data. In response to research question 3, we present two novel algorithms to perform efficient feature selection. We leverage the power of SNNs to advance the scalability of feature selection to high-dimensional datasets.

- **QuickSelection.** In Chapter 4, we proposed QuickSelection, the first method to exploit the power of sparse neural networks and dynamic sparse training to perform energy-efficient feature selection. QuickSelection blends the representation learning power of neural networks and dimensionality reduction to perform efficient and effective learning. It introduces neuron strength within sparse neural networks to determine feature importance. QuickSelection employs sparsely connected denoising autoencoders trained with the dynamic sparse training approach to derive the ranking of the features efficiently. This efficiency is shown with faster processing and reduced memory consumption. Through extensive experimental evaluation, we show that QuickSelection is the top performer in balancing accuracy and computational efficiency compared to neural

network-based and classical feature selection methods. QuickSelection's early detection of relevant features adds to its effectiveness. Our proposed method aligns with the growing emphasis on trustworthy AI that is both accurate and environmentally responsible. In an era of high-dimensional and complex datasets, QuickSelection addresses the need for cost-effective and environmentally responsible AI in these challenging environments.

- **NeuroFS.** Despite the promising results of QuickSelection which exploits the characteristics of a sparse neural network to perform efficient feature selection, it might suffer from slow convergence (due to random topology search) or subpar performance on very high-dimensional datasets (due to large search space). To address these challenges, in Chapter 5, we propose NeuroFS, a novel supervised feature selection method that leverages sparse neural networks with dynamic neuron pruning and regrowing during training. We introduce for the first time neuron evolution into the dynamics of the DST framework to gradually remove the unimportant features. In addition, we exploit gradient weight regrowth to speed up the training convergence. Comparative evaluations against various state-of-the-art feature selection methods demonstrate that NeuroFS outperforms its competitors in terms of classification accuracy. NeuroFS demonstrated robustness to topology changes, finding various topologies with close feature selection performance. Moreover, NeuroFS showcased superior computational efficiency compared to other methods like LassoNet, and STG that exploit MLP with sparsity in the input layer. Overall, this work contributes to the growing field of feature selection by presenting a resource-efficient and high-performance solution for addressing the challenges posed by high-dimensional data.

By addressing the research questions and providing innovative solutions to enhance the efficiency of deep neural networks through sparsity and feature selection, this thesis contributes to the development of cost-effective artificial neural networks. The proposed methods open up new avenues for research and pave the way for the practical deployment of deep learning models in various environments with limited computational and memory resources and high-dimensional data. This research contributes to making AI systems not only economically efficient but also environmentally responsible.

6.2 Insights for Practitioners

In this section, we provide practical guidelines for practitioners seeking to integrate the methodologies outlined in this thesis into their projects. We offer suggestions on when to utilize these methods over baselines and on which datasets to apply them to achieve better performance for the task at hand.

Dynamic Sparse Training In Chapter 2, we introduced a new dynamic sparse training model, named CTRE. We evaluated the performance of CTRE and the baselines on various classification tasks. As shown by the experiments on a noisy dataset, CTRE performs significantly better than the baselines in noisy environments. Therefore, CTRE can be a good choice for the classification of noisy datasets. Additionally, when analyzing the learning speed, CTRE showed fast adaptation to the data and gained a reasonable performance earlier than the baselines. In addition, we provided a truly sparse implementation of CTRE. As a result, in environments with limited computational resources, CTRE can be a viable choice.

Automatic Sparsity Tuning In Chapter 3, we presented a method for tuning the sparsity level when pruning neural networks, particularly when pruning transformers for time series forecasting. As observed in the experiments, transformers can be pruned to be more computationally efficient in all datasets, and even outperform the dense models in terms of prediction quality in simple tasks (such as the weather dataset in our experiments) where over-parameterization leads to over-fitting. However, when confronted with more intricate tasks such as traffic and electricity forecasting, overparameterization tends to yield more precise results. In comparison to alternative techniques for inducing sparsity, PALS emerges as a favorable option owing to its ability to automatically adjust the sparsity level without the need for a costly process for its tuning.

Feature Selection We proposed two feature selection methods for unsupervised and supervised feature selection, named QuickSelection (Chapter 4) and NeuroFS (Chapter 5), respectively.

- **QuickSelection.** Our experiments have demonstrated that QuickSelection stands out as a promising choice for low-dimensional datasets, primarily due to its efficiency and accuracy. When confronted with datasets with a

high number of samples, exemplified by MNIST in our experiments, QuickSelection continues to beat the baseline methods in terms of cumulative efficiency and accuracy score. Moreover, for high-dimensional datasets, statistical-based approaches offer superior computational efficiency but lag behind QuickSelection in terms of accuracy. Furthermore, QuickSelection emerges as the preferred option in noisy environments owing to its ability in identifying relevant features with high accuracy.

- **NeuroFS.** As shown in our experiments, NeuroFS is a favorable choice when performing supervised feature selection due to its superior performance, and low computational costs. It stands out overall as the best performer in terms of feature selection performance in low and high-dimensional datasets and can significantly reduce the memory and FLOPs count when compared to the deep learning-based competitors. However, we observed that NeuroFS can be sensitive to noise. Alternative feature selection methods in noisy environments are QuickSelection (for unsupervised tasks) and LassoNet (for supervised tasks).

6.3 Preliminary Impact

6.3.1 Energy efficiency

One of the main drawbacks of unstructured sparsity is the lack of support for sparse matrix operations on commodity hardware. Most existing approaches simulate sparsity by using a binary mask applied to the weights. Therefore, they fail to fully benefit from the real speed-up of sparse neural networks and energy consumption reduction. In this thesis, in Chapters 2 and 4, we implemented MLPs and Autoencoders using a truly sparse code to benefit from the real speedup and energy consumption reduction of sparse neural networks. Notably, in Appendix C.1.2, our research demonstrates significant power consumption reductions of up to an order of magnitude when applied to a high-dimensional dataset like GLA, compared to its direct competitor, CAE. Moreover, we showed the linear increase in running time for QuickSelection on CPU compared to the quadratic growth for CAE on GPU as the number of input features rises. These findings pave the way to more sustainable approaches in the field of deep learning.

6.3.2 QuickSelection

QuickSelection has laid the foundation for a novel research direction at the intersection of dynamic sparse training and feature selection, where performance and efficiency meet. In [SAPM22], we further explore the idea of neuron strength introduced in QuickSelection, by taking the sensitivity of the loss to the output neurons into account for computing input neuron importance. In our analysis, we show improved performance and speedup in convergence compared to the baselines. [GSD+23] explores the principle from QuickSelection and takes it to the world of Deep Reinforcement Learning, showing the effectiveness of dynamic sparse agents in noisy environments, their ability to focus on task-relevant features, and their rapid adaptability to new tasks. QuickSelection has also inspired a range of noteworthy student theses, among which a B.Sc. thesis won the best student thesis award at the University of Twente [KAM21] and an M.Sc. thesis has received cum laude distinction at the Eindhoven University of Technology.

6.4 Limitations

Although CeANNs can be a viable alternative for saving energy in low-resource devices and data centers and pave the way to achieving environmentally friendly AI systems, it is crucial to acknowledge their limitations. In this section, we summarize the current limitations of this research.

The proposed CeANNs leverage unstructured sparsity to reduce computational costs posed by over-parametrized models or high-dimensional data. However, they may not fully take advantage of their potential computational and memory benefits due to the lack of appropriate hardware support for unstructured sparsity for on-GPU training, as many other methods for obtaining sparse neural networks. As demonstrated in Chapter 4, we illustrated a significant increase in speed by implementing QuickSelection using sparse matrix operations on the CPU. However, for on-GPU processing, sparsity is simulated using a binary mask overweight, therefore, the potential gain is shown theoretically. This presents a formidable challenge addressing which requires a deep understanding of hardware architecture and a substantial research effort. Nonetheless, as the body of work concerning sparse neural networks continues to expand, there is increasing support for sparsity software and hardware development [GZYE20, CGG+21, ZLL+22, EDGS20, HSRN+19, WJH+18, ABB+19, LCC+21]. This ongoing development could lead to enhanced training and inference speeds,

as well as reduced memory requirements, thereby offering tangible benefits of CeANNs in real-world applications.

Optimizing the balance between computational efficiency and the desired performance is a very challenging task. There are instances where reducing computational costs may lead to slight performance degradation. While the derived sparsity in model and input features in this thesis can substantially reduce computational costs and energy consumption in low-resource devices and data centers, it is essential to recognize that this trade-off could raise concerns in situations where any reduction in accuracy might compromise user safety [CDP+19, MUL+20]. Therefore, a thorough examination and adjustment of the network's sparsity level and the dimension of selected features are necessary to ensure the optimal balance between energy efficiency and performance for specific applications.

6.5 Future Work

In this section, we suggest future directions for researchers to contribute to the advancement of CeANNs.

Feature Selection for Time Series Data. A promising future direction is the development of feature selection techniques tailored to handle very large time series datasets. With the proliferation of large time series data across domains such as finance, healthcare, and environmental monitoring, the need to efficiently extract relevant features from these massive and complex datasets has become increasingly critical [SYD19, DSSK19]. Existing feature selection methods mostly are designed for tabular datasets and may not adequately address the unique challenges posed by time series data, including temporal dependencies and noisy observations. Future research can focus on designing algorithms that select important features from time series data while reducing computational overhead. These techniques should be designed to enhance the interpretability, accuracy, and scalability of time series analysis.

Enhancing Explainability in CeANNs. The integration of explainable AI (XAI) into this research represents a promising future direction, with the potential to improve not only the efficiency of artificial neural networks (ANNs) but also their transparency and interpretability. The developed feature selection models significantly contribute to enhancing interpretability across various domains by

providing valuable insights into the datasets and how neural network models learn the importance of input features in the initial layer. By further developing methods to make CeANNs models more explainable and interpretable in subsequent layers and decision-making processes (e.g., by studying hidden layer neuron importance), the research can contribute to comprehending the reasons and mechanisms governing ANNs' decision-making, which is crucial for real-world applications and regulatory compliance [KKS+19, PNBAJ+23, MUL+20]. This approach aligns with the growing demand for AI models that are not just efficient but also transparent and accountable, particularly in domains like healthcare, finance, and autonomous systems.

Efficient Architecture Search. One interesting future direction is to efficiently optimize neural network architectures. Currently, the developed methods in this thesis focus on fine-tuning sparse connections within fixed network capacity and/or adjusting sparsity levels. The resource-intensive nature of Neural Architecture Search (NAS) demonstrates the need for more efficient approaches [SGM20]. Therefore, an interesting future study is how such architectural optimization can be integrated effectively within cost-efficient artificial neural networks. This investigation has the potential to mitigate the computational burden of architectural exploration while still delivering tailored solutions for diverse applications.

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. (Cited on page 29.)

[ABB+19] Mike Ashby, Christiaan Baaij, Peter Baldwin, Martijn Bastiaan, Oliver Bunting, Aiken Cairncross, Christopher Chalmers, Liz Corrigan, Sam Davis, Nathan van Doorn, et al. Exploiting unstructured sparsity on next-generation datacenter hardware, 2019. (Cited on page 125.)

[ABGM14] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In International conference on machine learning, pages 584–592. PMLR, 2014. (Cited on page 21.)

[AGO+13] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In Esann, 2013. (Cited on page 80.)

[AMHH15] Jun Chin Ang, Andri Mirzal, Habibollah Haron, and Haza Nuzly Abdull Hamed. 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. (Cited on page 68.)

[APL+22] Zahra Atashgahi, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. A brain-inspired algorithm for training highly sparse neural networks. Machine Learning, 111(12):4411–4452, 2022. (Cited on pages 10, 51, and 100.)

[APVM23] Zahra Atashgahi, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu. Adaptive sparsity level during training for efficient time series forecasting with transformers. arXiv preprint arXiv:2305.18382, 2023. (Cited on page 11.)

[ASL+18] Amirali Aghazadeh, Ryan Spring, Daniel Lejeune, Gautam Dasarathy, Anshumali Shrivastava, et al. Mission: Ultra large-scale feature selection using count-sketches. In International Conference on Machine Learning, pages 80–88, 2018. (Cited on page 69.)

[ASvdL+22] Zahra Atashgahi, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. Machine Learning, 111:377–414, 2022. (Cited on pages 12, 16, 20, 55, 96, 98, 99, 101, 107, 108, and 189.)

[ATZ+22] Milad Alizadeh, Shyam A. Tailor, Luisa M Zintgraf, Joost van Amersfoort, Sebastian Farquhar, Nicholas Donald Lane, and Yarin Gal. Prospect pruning: Finding trainable weights at initialization using meta-gradients. In International Conference on Learning Representations, 2022. (Cited on page 6.)

[AZK+23] Zahra Atashgahi, Xuhao Zhang, Neil Kichler, Shiwei Liu, Lu Yin, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu. Supervised feature selection with neuron evolution in sparse neural networks. Transactions on Machine Learning Research, 2023. (Cited on page 12.)

[AZLS19] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 242–252. PMLR, 09–15 Jun 2019. (Cited on page 1.)

[Bal12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Proceedings of ICML workshop on unsupervised and transfer learning, pages 37–49, 2012. (Cited on page 72.)

[Bat94] Roberto Battiti. Using mutual information for selecting features in supervised neural net learning. IEEE Transactions on neural networks, 5(4):537–550, 1994. (Cited on pages 7, 8, and 98.)

[BAZ19] Muhammed Fatih Balın, Abubakar Abid, and James Zou. Concrete autoencoders: Differentiable feature selection and reconstruction. In International Conference on Machine Learning, pages 444–453, 2019. (Cited on pages 8, 72, 96, and 98.)

[BBPSV04] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. Proceedings of the national academy of sciences, 101(11):3747–3752, 2004. (Cited on page 76.)

[BCSMAB15] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. Feature selection for high-dimensional data. Springer, 2015. (Cited on page 69.)

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013. (Cited on page 71.)

[BGMSS18] Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz. SGD learns over-parameterized networks that provably generalize on linearly separable data. In International Conference on Learning Representations, 2018. (Cited on page 1.)

[BJRL15] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. Time series analysis: forecasting and control. John Wiley & Sons, 2015. (Cited on page 53.)

[BKML18] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In International Conference on Learning Representations, 2018. (Cited on pages 7, 17, 20, 74, and 99.)

[BLN+23] Simone Borsci, Ville V Lehtola, Francesco Nex, Michael Ying Yang, Ellen-Wien Augustijn, Leila Bagheriye, Christoph Brune, Ourania Kounadi, Jamy Li, Joao Moreira, et al. Embedding artificial intelligence in society: looking beyond the eu ai master plan using the culture cycle. AI & society, 38(4):1465–1484, 2023. (Cited on page 2.)

[BM13] Julian Bishop and Risto Miikkulainen. Evolutionary feature evaluation for online reinforcement learning. In 2013 IEEE Conference on Computational Inteligence in Games (CIG), pages 1–8. IEEE, 2013. (Cited on page 71.)

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. (Cited on pages 1 and 16.)

[BPR+19] David D. Bourgin, Joshua C. Peterson, Daniel Reichman, Stuart J. Russell, and Thomas L. Griffiths. Cognitive model priors for predicting human decisions. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 5133–5141, Long Beach, California, USA, 09–15 Jun 2019. PMLR. (Cited on page 73.)

[BSR+18] Sergey Bartunov, Adam Santoro, Blake A Richards, Luke Marris, Geoffrey E Hinton, and Timothy P Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, pages 9390–9400, 2018. (Cited on page 46.)

[C+15] François Chollet et al. Keras. *https://keras.io*, 2015. (Cited on pages 108 and 202.)

[CCG+21] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. Advances in Neural Information Processing Systems, 34:19974–19988, 2021. (Cited on page 52.)

[CDP+19] Robert Challen, Joshua Denny, Martin Pitt, Luke Gompels, Tom Edwards, and Krasimira Tsaneva-Atanasova. Artificial intelligence, bias and clinical safety. BMJ Quality & Safety, 28(3):231–237, 2019. (Cited on page 126.)

[CFC+20] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 15834–15846. Curran Associates, Inc., 2020. (Cited on page 52.)

[CGG+21] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. IEEE Micro, 41(2):29–35, 2021. (Cited on page 125.)

[CMP21] Selima Curci, Decebal Constantin Mocanu, and Mykola Pechenizkiyi. Truly sparse neural networks at scale. arXiv preprint arXiv:2102.01732, 2021. (Cited on page 189.)

[COO+22] Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza, Max Mergenthaler, and Artur Dubrawski. N-hits: Neural hierarchical interpolation for time series forecasting. arXiv preprint arXiv:2201.12886, 2022. (Cited on page 53.)

[CS14] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. Computers & Electrical Engineering, 40(1):16–28, 2014. (Cited on pages 7, 68, 70, 96, and 98.)

[CS15] B Chandra and Rajesh K Sharma. Exploring autoencoders for unsupervised feature selection. In 2015 International Joint Conference on Neural Networks (IJCNN), pages 1–6. IEEE, 2015. (Cited on pages 8, 96, and 98.)

[CYM+23] Yubei Chen, Zeyu Yun, Yi Ma, Bruno Olshausen, and Yann LeCun. Minimalistic unsupervised representation learning with the sparse manifold transform. In The Eleventh International Conference on Learning Representations, 2023. (Cited on page 4.)

[CZH10] Deng Cai, Chiyuan Zhang, and Xiaofei He. Unsupervised feature selection for multi-cluster data. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 333–342. ACM, 2010. (Cited on page 71.)

[CZpw+22] Tianlong Chen, Zhenyu Zhang, pengjun wang, Santosh Balachandra, Haoyu Ma, Zehao Wang, and Zhangyang Wang. Sparsity winning twice: Better robust generalization from more efficient training. In International Conference on Learning Representations, 2022. (Cited on page 4.)

[DA22] Pradip Dhal and Chandrashekhar Azad. A comprehensive survey on feature selection in the various fields of machine learning. Applied Intelligence, pages 1–39, 2022. (Cited on page 8.)

[DB04] Jennifer G Dy and Carla E Brodley. Feature selection for unsupervised learning. Journal of machine learning research, 5(Aug):845–889, 2004. (Cited on page 68.)

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020. (Cited on pages 48 and 52.)

[DGO+22] Anastasia S. D. Dietrich, Frithjof Gressmann, Douglas Orr, Ivan Chelombiev, Daniel Justus, and Carlo Luschi. Towards structured dynamic sparse pre-training of BERT, 2022. (Cited on page 52.)

[dJSB+21] Pau de Jorge, Amartya Sanyal, Harkirat Behl, Philip Torr, Grégory Rogez, and Puneet K. Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In International Conference on Learning Representations, 2021. (Cited on pages 5 and 20.)

[DS19] Guillaume Doquet and Michèle Sebag. Agnostic feature selection. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 343–358. Springer, 2019. (Cited on pages 8, 71, 96, and 98.)

[DSSK19] Sabyasachi Dash, Sushil Kumar Shakyawar, Mohit Sharma, and Sandeep Kaushik. Big data in healthcare: management, analysis and future prospects. Journal of big data, 6(1):1–25, 2019. (Cited on page 126.)

[DYJ19] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers, 68(10):1487–1497, 2019. (Cited on pages 7, 21, and 100.)

[DZ19] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840, 2019. (Cited on pages 18, 20, 74, and 100.)

[DZPS19] Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In International Conference on Learning Representations, 2019. (Cited on page 1.)

[EDGS20] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 14629–14638, 2020. (Cited on page 125.)

[EGM+20] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In International Conference on Machine Learning, pages 2943–2952. PMLR, 2020. (Cited on pages xxii, 3, 7, 17, 20, 29, 39, 49, 51, 52, 60, 63, 74, 100, 101, 105, 108, 115, 116, and 215.)

[EIKD22] Utku Evci, Yani Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 6577–6586, 2022. (Cited on pages 100 and 101.)

[EKN+17] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. nature, 542(7639):115–118, 2017. (Cited on page 1.)

[FC91] Mark Fanty and Ronald Cole. Spoken letter recognition. In Advances in Neural Information Processing Systems, pages 220–226, 1991. (Cited on pages 30 and 80.)

[FC19] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In International Conference on Learning Representations, 2019. (Cited on pages 5, 16, 19, 49, 51, 99, 100, and 205.)

[FDJ19] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. Advances in neural information processing systems, 32, 2019. (Cited on page 53.)

[FGK13] Ahmed K Farahat, Ali Ghodsi, and Mohamed S Kamel. Efficient greedy feature selection for unsupervised learning. Knowledge and information systems, 35(2):285–310, 2013. (Cited on page 71.)

[Fri08] Karl Friston. Hierarchical models in the brain. PLoS computational biology, 4(11):e1000211, 2008. (Cited on page 16.)

[FST+22] Takashi Furuya, Kazuma Suetake, Koichi Taniguchi, Hiroyuki Kusumoto, Ryuji Saiin, and Tomohiro Daimon. Spectral pruning for recurrent neural networks. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, Proceedings of The 25th International Conference on Artificial Intelligence and Statistics, volume 151 of Proceedings of Machine Learning Research, pages 3458–3482. PMLR, 28–30 Mar 2022. (Cited on page 52.)

[Gam17] John Cristian Borges Gamboa. Deep learning for time-series analysis. arXiv preprint arXiv:1701.01887, 2017. (Cited on page 53.)

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016. (Cited on pages 1 and 98.)

[GCL+21] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Hassan Sajjad, Preslav Nakov, Deming Chen, and Marianne Winslett. Compressing large-scale transformer-based models: A case study on bert. Transactions of the Association for Computational Linguistics, 9:1061–1080, 2021. (Cited on page 52.)

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. Journal of machine learning research, 3(Mar):1157–1182, 2003. (Cited on pages 7 and 98.)

[GEH19] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574, 2019. (Cited on pages 20 and 99.)

[GEN+18] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1586–1595, 2018. (Cited on page 20.)

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. Machine learning, 63(1):3–42, 2006. (Cited on page 81.)

[GGNZ08] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. Feature extraction: foundations and applications, volume 207. Springer, 2008. (Cited on pages 30 and 80.)

[GLH11] Quanquan Gu, Zhenhui Li, and Jiawei Han. Generalized fisher score for feature selection. In 27th Conference on Uncertainty in Artificial Intelligence, UAI 2011, pages 266–273, 2011. (Cited on page 107.)

[GMB23] Advait Harshal Gadhikar, Sohom Mukherjee, and Rebekka Burkholz. Why random pruning is all we need to start sparse. In International Conference on Machine Learning, pages 10542–10570. PMLR, 2023. (Cited on page 74.)

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing, pages 6645–6649. Ieee, 2013. (Cited on page 16.)

[GRKB21] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. arXiv preprint arXiv:2106.11959, 2021. (Cited on page 18.)

[Gro20] AI High-Level Expert Group. Assessment list for trustworthy artificial intelligence (ALTAI) for self-assessment, 2020. (Cited on pages 2, 16, and 68.)

[GS21] Lukas Galke and Ansgar Scherp. Forget me not: A gentle reminder to mind the simple multi-layer perceptron baseline for text classification. arXiv preprint arXiv:2109.03777, 2021. (Cited on page 19.)

[GSD+23] Bram Grooten, Ghada Sokar, Shibhansh Dohare, Elena Mocanu, Matthew E. Taylor, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Automatic noise filtering with dynamic sparse training in deep reinforcement learning. In Proceedings of the 2023 International Conference on Autonomous Agents and

Multiagent Systems, AAMAS '23, page 1932–1941, Richland, SC, 2023. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on page 125.)

[GWBV02] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. Machine learning, 46(1):389–422, 2002. (Cited on pages 8 and 98.)

[GYC16] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, page 1387–1395, Red Hook, NY, USA, 2016. Curran Associates Inc. (Cited on page 20.)

[GZYE20] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–14. IEEE, 2020. (Cited on page 125.)

[HA18] Rob J Hyndman and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2018. (Cited on page 53.)

[HABN+21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. J. Mach. Learn. Res., 22(1), jan 2021. (Cited on pages 1, 2, 3, 4, 7, 16, 49, 52, 56, 96, 99, 100, and 189.)

[HCN06] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. In Advances in neural information processing systems, pages 507–514, 2006. (Cited on pages 70 and 98.)

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine, 29(6):82–97, 2012. (Cited on page 1.)

[Heb05] Donald Olding Hebb. The organization of behavior: A neuropsychological theory. Psychology Press, 2005. (Cited on pages 18 and 21.)

[HKP+12] Jiawei Han, Micheline Kamber, Jian Pei, et al. Getting to know your data. In Data mining, pages 39–82. Elsevier Amsterdam, Netherlands, 2012. (Cited on page 21.)

[HLW16] Rob J Hyndman, Alan J Lee, and Earo Wang. Fast computation of reconciled forecasts for hierarchical and grouped time series. Computational statistics & data analysis, 97:16–32, 2016. (Cited on pages 9 and 48.)

[HNA+17] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409, 2017. (Cited on pages 16 and 96.)

[Hoo21] Sara Hooker. The hardware lottery. Communications of the ACM, 64(12):58–65, 2021. (Cited on pages 3 and 115.)

[HPN+17] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, Bryan Catanzaro, and William J. Dally. DSD: Dense-sparse-dense training for deep neural networks. In International Conference on Learning Representations, 2017. (Cited on page 57.)

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. Advances in neural information processing systems, 28, 2015. (Cited on pages 4, 5, 16, 19, 49, 51, 72, and 99.)

[HS93] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In Advances in neural information processing systems, pages 164–171, 1993. (Cited on pages 5, 19, 72, and 99.)

[HSRN+19] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P Sadayappan. Adaptive sparse tiling for sparse matrix multiplication. In Proceedings of the 24th Symposium

on Principles and Practice of Parallel Programming, pages 300–314, 2019. (Cited on page 125.)

[HWZ+18] Kai Han, Yunhe Wang, Chao Zhang, Chao Li, and Chao Xu. Autoencoder inspired unsupervised feature selection. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 2941–2945. IEEE, 2018. (Cited on pages 8, 71, 96, and 98.)

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016. (Cited on page 1.)

[Jak05] Aleks Jakulin. Machine learning based on attribute interactions. PhD thesis, University of Ljubljana, 2005. (Cited on page 107.)

[JOP01] Eric Jones, Travis Oliphant, and Pearu Peterson. Scipy: Open source scientific tools for python, 2001. (Cited on page 79.)

[JPM+22] Xiaoyong Jin, Youngsuk Park, Danielle Maddix, Hao Wang, and Yuyang Wang. Domain adaptation for time series forecasting via attention sharing. In International Conference on Machine Learning, pages 10280–10297. PMLR, 2022. (Cited on page 53.)

[JPR+20] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. Advances in Neural Information Processing Systems, 33:20744–20754, 2020. (Cited on pages 7, 17, 21, 51, and 100.)

[JYP+17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th annual international symposium on computer architecture, pages 1–12, 2017. (Cited on page 19.)

[JZR+19] LIU Junjie, XU Zhe, SHI Runbin, Ray CC Cheung, and Hayden KH So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In International Conference on Learning Representations, 2019. (Cited on page 20.)

[KAM21] Neil Kichler, Zahra Atashgahi, and Decebal Constantin Mocanu. Robustness of sparse mlps for supervised feature selection. In Sparsity in Neural Networks: Advancing Understanding and Practice 2021, 2021. (Cited on page 125.)

[KBCD14] Taras Kowaliw, Nicolas Bredeche, Sylvain Chevallier, and René Doursat. Artificial neurogenesis: An introduction and selective review. Growing Adaptive Machines, pages 1–60, 2014. (Cited on page 101.)

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009. (Cited on page 31.)

[KJ97] Ron Kohavi and George H John. Wrappers for feature subset selection. Artificial intelligence, 97(1-2):273–324, 1997. (Cited on pages 7, 71, and 98.)

[KKL20] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In International Conference on Learning Representations, 2020. (Cited on page 53.)

[KKOI22] Kazuki Koyama, Keisuke Kiritoshi, Tomomi Okawachi, and Tomonori Izumitani. Effective nonlinear feature selection method based on hsic lasso and with variational inference. In International Conference on Artificial Intelligence and Statistics, pages 10407–10421. PMLR, 2022. (Cited on page 215.)

[KKS+19] Christopher J Kelly, Alan Karthikesalingam, Mustafa Suleyman, Greg Corrado, and Dominic King. Key challenges for delivering clinical impact with artificial intelligence. BMC medicine, 17:1–9, 2019. (Cited on page 127.)

[KM98] Taskin Kavzoglu and Paul M Mather. Assessing artificial neural network pruning algorithms. In Proceedings of the 24th Annual Conference and Exhibition of the Remote Sensing Society, pages 9–11, 1998. (Cited on page 93.)

[KMST15] Eduard Kuriscak, Petr Marsalek, Julius Stroffek, and Peter G Toth. Biological context of hebb learning in artificial neural networks, a review. Neurocomputing, 152:27–35, 2015. (Cited on page 21.)

[KR19] Jeremy Kepner and Ryan Robinett. Radix-net: Structured sparse matrices for deep neural networks. In 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 268–274. IEEE, 2019. (Cited on pages 17 and 20.)

[KRS+20] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In Hal Daumé III and Aarti Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 5544–5555. PMLR, 13–18 Jul 2020. (Cited on page 20.)

[KSBM01] S. Sathiya Keerthi, Shirish Krishnaj Shevade, Chiranjib Bhattacharyya, and Karuturi Radha Krishna Murthy. Improvements to platt's smo algorithm for svm classifier design. Neural computation, 13(3):637–649, 2001. (Cited on page 109.)

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 2012. (Cited on page 1.)

[KW09] Mark Kroon and Shimon Whiteson. Automatic feature selection for model-based reinforcement learning in factored mdps. In 2009 International Conference on Machine Learning and Applications, pages 324–330. IEEE, 2009. (Cited on page 71.)

[KYGJ19] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In IJCAI, pages 2725–2732, 2019. (Cited on page 52.)

[Lan95] Ken Lang. Newsweeder: Learning to filter netnews. In Machine Learning Proceedings 1995, pages 331–339. Elsevier, 1995. (Cited on pages 31 and 80.)

[LAT19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In International Conference on Learning Representations, 2019. (Cited on pages 5, 16, 20, 28, 51, 72, and 99.)

[LCC+21] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. Advances in Neural Information Processing Systems, 34:9908–9922, 2021. (Cited on pages xxii, xxv, 5, 20, 29, 49, 51, 52, 54, 55, 56, 60, 63, 100, 125, 177, 179, and 189.)

[LCW16] Yifeng Li, Chih-Yu Chen, and Wyeth W Wasserman. Deep feature selection: theory and application to identify enhancers and promoters. Journal of Computational Biology, 23(5):322–336, 2016. (Cited on page 71.)

[LCW+18] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. ACM Computing Surveys (CSUR), 50(6):94, 2018. (Cited on pages 8, 71, 79, 81, 96, and 108.)

[LCWE06] Thomas Navin Lal, Olivier Chapelle, Jason Weston, and André Elisseeff. Embedded methods. In Feature extraction, pages 137–165. Springer, 2006. (Cited on page 71.)

[LCYL18] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In The 41st international ACM SIGIR conference on research & development in information retrieval, pages 95–104, 2018. (Cited on pages 53 and 175.)

[LDS90] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In Advances in neural information processing systems, pages 598–605, 1990. (Cited on pages 4, 5, 16, 19, 72, and 99.)

[LeC98] Yann LeCun. The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/, 1998. (Cited on pages 30 and 80.)

[LFLN18] Yang Lu, Yingying Fan, Jinchi Lv, and William Stafford Noble. Deeppink: reproducible feature selection in deep neural networks. In Advances in Neural Information Processing Systems, pages 8676–8686, 2018. (Cited on pages 8, 71, 96, and 98.)

[LGM17] Jia Liu, Maoguo Gong, and Qiguang Miao. Modeling hebb learning rule for unsupervised learning. In IJCAI, pages 2315–2321, 2017. (Cited on page 21.)

[LGM+20] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8018–8027, 2020. (Cited on pages 5 and 19.)

[LH13] Baoli Li and Liping Han. Distance weighted cosine similarity measure for text classification. In International conference on intelligent data engineering and automated learning, pages 611–618. Springer, 2013. (Cited on page 22.)

[LH15] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3367–3375, 2015. (Cited on page 16.)

[LJX+19] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019. (Cited on page 53.)

[LLWD22] Yan Li, Xinjiang Lu, Yaqing Wang, and Dejing Dou. Generative time series forecasting with diffusion, denoise, and disentanglement. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. (Cited on page 53.)

[LM98] Huan Liu and Hiroshi Motoda. Feature extraction, construction and selection: A data mining perspective, volume 453. Springer Science & Business Media, 1998. (Cited on page 68.)

[LMPP21] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. In Marina Meila

and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 6893–6904. PMLR, 18–24 Jul 2021. (Cited on pages 20 and 52.)

[LRAT21] Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. Journal of Machine Learning Research, 22(127):1–29, 2021. (Cited on pages 8, 96, 98, 107, and 108.)

[LS+96] Huan Liu, Rudy Setiono, et al. A probabilistic approach to feature selection-a filter solution. In ICML, volume 96, pages 319–327. Citeseer, 1996. (Cited on pages 7 and 98.)

[LT06] Dahua Lin and Xiaoou Tang. Conditional infomax learning: An integrated framework for feature extraction and fusion. In European conference on computer vision, pages 68–82. Springer, 2006. (Cited on page 107.)

[LTM+12] Cosmin Lazar, Jonatan Taminau, Stijn Meganck, David Steenhoff, Alain Coletta, Colin Molter, Virginie de Schaetzen, Robin Duque, Hugues Bersini, and Ann Nowe. A survey on filter techniques for feature selection in gene expression microarray analysis. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 9(4):1106–1119, 2012. (Cited on page 70.)

[LvdLY+20] Shiwei Liu, Tim van der Lee, Anil Yaman, Zahra Atashgahi, Davide Ferraro, Ghada Sokar, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Topological insights into sparse neural networks. In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) 2020., pages 2006–14085, 2020. (Cited on pages 20, 76, and 113.)

[LW+02] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. R news, 2(3):18–22, 2002. (Cited on pages 81 and 207.)

[LW19] Congcong Liu and Huaming Wu. Channel pruning based on mean gradient for accelerating convolutional neural networks. Signal Processing, 156:84–91, 2019. (Cited on pages 5 and 19.)

[LW23] Shiwei Liu and Zhangyang Wang. Ten lessons we have learned in the new" sparseland": A short handbook for sparse neural network researchers. arXiv preprint arXiv:2302.02596, 2023. (Cited on pages 4 and 189.)

[LWK18] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In International Conference on Learning Representations, 2018. (Cited on pages 5, 16, 20, and 51.)

[LWS+20] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In Hal Daumé III and Aarti Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 5958–5968. PMLR, 13–18 Jul 2020. (Cited on page 52.)

[LWWL22] Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Rethinking the stationarity in time series forecasting. arXiv preprint arXiv:2205.14415, 2022. (Cited on pages 48, 50, 53, 60, and 189.)

[LYL+21] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In International Conference on Learning Representations, 2021. (Cited on page 53.)

[LYMP21] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In Marina Meila and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 6989–7000. PMLR, 18–24 Jul 2021. (Cited on pages 17, 20, 29, 51, 55, and 100.)

[LZ21] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. Philosophical Transactions of the Royal

Society A, 379(2194):20200209, 2021. (Cited on pages 48 and 53.)

[LZX+18] Chunjie Luo, Jianfeng Zhan, Xiaohe Xue, Lei Wang, Rui Ren, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. In International Conference on Artificial Neural Networks, pages 382–391. Springer, 2018. (Cited on page 22.)

[MAV17] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In International Conference on Machine Learning, pages 2498–2507. PMLR, 2017. (Cited on page 20.)

[MHP+17] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the granularity of sparsity in convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 13–20, 2017. (Cited on page 4.)

[MLN19] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? Advances in neural information processing systems, 32, 2019. (Cited on page 52.)

[MMN+16] Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. Machine Learning, 104(2-3):243–270, 2016. (Cited on pages 6, 17, 20, 51, 73, and 99.)

[MMP+21] Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H. Nguyen, Madeleine Gibescu, Damien Ernst, and Zita A. Vale. Sparse training theory for scalable and efficient agents. In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '21, page 34–38, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on pages 4, 16, 17, 51, and 99.)

[MMS+18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive

sparse connectivity inspired by network science. Nature communications, 9(1):1–12, 2018. (Cited on pages xxii, 6, 7, 17, 20, 29, 39, 49, 50, 51, 55, 56, 73, 74, 75, 99, 100, 103, and 108.)

[MMT+19] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019. (Cited on pages 5, 19, and 99.)

[MN16] Jianyu Miao and Lingfeng Niu. A survey on feature selection. Procedia Computer Science, 91:919–926, 2016. (Cited on page 68.)

[MQS+22] Xiaolong Ma, Minghai Qin, Fei Sun, Zejiang Hou, Kun Yuan, Yi Xu, Yanzhi Wang, Yen-Kuang Chen, Rong Jin, and Yuan Xie. Effective model sparsification by scheduled grow-and-prune methods. In International Conference on Learning Representations, 2022. (Cited on page 57.)

[MTK+17] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In International Conference on Learning Representations, 2017. (Cited on pages 5, 19, and 99.)

[MUL+20] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. IEEE Transactions on Intelligent Transportation Systems, 22(7):4316–4336, 2020. (Cited on pages 126 and 127.)

[MW19] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 4646–4655. PMLR, 09–15 Jun 2019. (Cited on pages 7, 17, 20, 74, and 100.)

[MWHN18] Iacopo Masi, Yue Wu, Tal Hassner, and Prem Natarajan. Deep face recognition: A survey. In 2018 31st SIBGRAPI conference

on graphics, patterns and images (SIBGRAPI), pages 471–478. IEEE, 2018. (Cited on page 16.)

[NB10] Hieu V Nguyen and Li Bai. Cosine similarity metric learning for face verification. In Asian conference on computer vision, pages 709–720. Springer, 2010. (Cited on page 22.)

[NHCD10] Feiping Nie, Heng Huang, Xiao Cai, and Chris Ding. Efficient and robust feature selection via joint l2, 1-norms minimization. Advances in neural information processing systems, 23, 2010. (Cited on page 107.)

[NLB+19] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In International Conference on Learning Representations, 2019. (Cited on pages 1 and 16.)

[NNM+96] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-20), 1996. (Cited on page 79.)

[NSvKS23] Meike Nauta, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. Pip-net: Patch-based intuitive prototypes for interpretable image classification. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2744–2753, 2023. (Cited on page 4.)

[OCCB19] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In International Conference on Learning Representations, 2019. (Cited on page 53.)

[PD21] Shreyas Malakarjun Patil and Constantine Dovrolis. Phew: Constructing sparse networks that learn fast and generalize well without training data. In International Conference on Machine Learning, pages 8432–8442. PMLR, 2021. (Cited on page 6.)

[PLD05] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence, 27(8):1226–1238, 2005. (Cited on pages 8 and 98.)

[PMB19] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. arXiv preprint arXiv:1909.06312, 2019. (Cited on page 18.)

[PMLL21] Roman Pogodin, Yash Mehta, Timothy P Lillicrap, and Peter E Latham. Towards biologically plausible convolutional networks. arXiv preprint arXiv:2106.13031, 2021. (Cited on page 46.)

[PNBAJ+23] Euclides Carlos Pinto Neto, Derick Moreira Baum, Jorge Rady de Almeida Jr, João Batista Camargo Jr, and Paulo Sergio Cugnasca. Deep learning in air traffic management (atm): A survey on applications, opportunities, and open challenges. Aerospace, 10(4):358, 2023. (Cited on page 127.)

[PRR20] Sai Prasanna, Anna Rogers, and Anna Rumshisky. When bert plays the lottery, all tickets are winning. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020. (Cited on page 52.)

[QSC+17] Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In Proceedings of the 26th International Joint Conference on Artificial Intelligence, pages 2627–2633, 2017. (Cited on page 53.)

[RA20] Md Aamir Raihan and Tor M Aamodt. Sparse weight activation training. arXiv preprint arXiv:2001.01969, 2020. (Cited on page 20.)

[RCM+12] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 262–270, 2012. (Cited on pages 9 and 48.)

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 779–788, 2016. (Cited on page 1.)

[SAPM22] Ghada Sokar, Zahra Atashgahi, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Where to pay attention in sparse training for feature selection? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, Advances in Neural Information Processing Systems, volume 35, pages 1627–1642. Curran Associates, Inc., 2022. (Cited on page 125.)

[SB16] Benjamin Scellier and Yoshua Bengio. Towards a biologically plausible backprop. arXiv preprint arXiv:1602.05179, 914, 2016. (Cited on page 21.)

[SBS+07] Avrum Spira, Jennifer E Beane, Vishal Shah, Katrina Steiling, Gang Liu, Frank Schembri, Sean Gilman, Yves-Martine Dumas, Paul Calner, Paola Sebastiani, et al. Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. Nature medicine, 13(3):361–366, 2007. (Cited on page 80.)

[Sch21] Thomas Schumacher. Livewired neural networks: Making neurons that fire together wire together. arXiv preprint arXiv:2105.08111, 2021. (Cited on page 24.)

[SFGJ20] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting, 36(3):1181–1191, 2020. (Cited on page 53.)

[SGGAP14] Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. Computación y Sistemas, 18(3):491–504, 2014. (Cited on page 22.)

[SGM20] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 13693–13696, 2020. (Cited on pages 1, 48, 119, and 127.)

[SHBB22] Georg Stefan Schlake, Jan David Hüwel, Fabian Berns, and Christian Beecks. Evaluating the lottery ticket hypothesis to sparsify neural networks for time series classification. In 2022 IEEE

38th International Conference on Data Engineering Workshops (ICDEW), pages 70–73, 2022. (Cited on page 52.)

[SHK23] Johannes Schmidt-Hieber and Wouter M Koolen. Hebbian learning inspired estimation of the linear regression parameters from queries. arXiv preprint arXiv:2311.03483, 2023. (Cited on page 21.)

[SHS+06] Lixin Sun, Ai-Min Hui, Qin Su, Alexander Vortmeyer, Yuri Kotliarov, Sandra Pastorino, Antonino Passaniti, Jayant Menon, Jennifer Walling, Rolando Bailey, et al. Neuronal and glioma-derived stem cell factor induces angiogenesis within the brain. Cancer cell, 9(4):287–300, 2006. (Cited on page 80.)

[SL97] Rudy Setiono and Huan Liu. Neural-network feature selector. IEEE transactions on neural networks, 8(3):654–662, 1997. (Cited on pages 8 and 98.)

[SL22] Yiyou Sun and Yixuan Li. Dice: Leveraging sparsification for out-of-distribution detection. In European Conference on Computer Vision, pages 691–708. Springer, 2022. (Cited on page 4.)

[SLSK22] Uri Shaham, Ofir Lindenbaum, Jonathan Svirsky, and Yuval Kluger. Deep unsupervised feature selection by discarding nuisance and correlated features. Neural Networks, 152:34–43, 2022. (Cited on page 96.)

[SM02] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2):99–127, 2002. (Cited on page 101.)

[SMM+21] Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. arXiv preprint arXiv:2106.04217, 2021. (Cited on pages 3, 51, and 115.)

[SRC+21] Cem Subakan, Mirco Ravanelli, Samuele Cornell, Mirko Bronzi, and Jianyuan Zhong. Attention is all you need in speech separation. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 21–25. IEEE, 2021. (Cited on page 48.)

[SSGC17] Razieh Sheikhpour, Mehdi Agha Sarram, Sajjad Gharaghani, and Mohammad Ali Zare Chahooki. A survey on semi-supervised feature selection methods. Pattern Recognition, 64:141–158, 2017. (Cited on page 68.)

[SSM20] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 11380–11390. Curran Associates, Inc., 2020. (Cited on page 20.)

[SWT16] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Sparsifying neural network connections for face recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4856–4864, 2016. (Cited on page 21.)

[SY20] Dinesh Singh and Makoto Yamada. Fsnet: Feature selection network on high-dimensional biological data. arXiv preprint arXiv:2001.08322, 2020. (Cited on pages 72, 96, and 98.)

[SYD19] Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. Advances in neural information processing systems, 32, 2019. (Cited on page 126.)

[TDBM22] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. ACM Computing Surveys, 55(6):1–28, 2022. (Cited on page 52.)

[THA+18] Thiyanga S Talagala, Rob J Hyndman, George Athanasopoulos, et al. Meta-learning how to forecast time series. Monash Econometrics and Business Statistics Working Papers, 6(18):16, 2018. (Cited on pages 9 and 48.)

[THK+21] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601, 2021. (Cited on page 19.)

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1):267–288, 1996. (Cited on page 96.)

[TKYG20] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. Advances in neural information processing systems, 33:6377–6389, 2020. (Cited on pages 5 and 20.)

[TTW14] Mingkui Tan, Ivor W Tsang, and Li Wang. Towards ultrahigh dimensional feature selection for big data. Journal of Machine Learning Research, 2014. (Cited on page 69.)

[VDMPVdH09] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. J Mach Learn Res, 10(66-71):13, 2009. (Cited on page 68.)

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM, 2008. (Cited on page 72.)

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017. (Cited on pages 1, 48, 50, and 60.)

[VTCZ+22] Mukund Varma T, Xuxi Chen, Zhenyu Zhang, Tianlong Chen, Subhashini Venugopalan, and Zhangyang Wang. Sparse winning tickets are data-efficient image recognizers. Advances in Neural Information Processing Systems, 35:4652–4666, 2022. (Cited on page 4.)

[WC20] Maksymilian Wojtas and Ke Chen. Feature importance ranking for deep learning. Advances in Neural Information Processing Systems, 33:5105–5114, 2020. (Cited on pages 8, 96, and 98.)

[WDS+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 conference on empirical methods in natural language

processing: system demonstrations, pages 38–45, 2020. (Cited on page 48.)

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3):37–52, 1987. (Cited on page 202.)

[WGFZ19] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In International Conference on Machine Learning, pages 6566–6575. PMLR, 2019. (Cited on pages 5 and 19.)

[WHL+22] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. arXiv preprint arXiv:2210.02186, 2022. (Cited on pages 48 and 184.)

[WJH+18] Peiqi Wang, Yu Ji, Chi Hong, Yongqiang Lyu, Dongsheng Wang, and Yuan Xie. Snrram: An efficient sparse neural network computation architecture based on resistive random-access memory. In Proceedings of the 55th Annual Design Automation Conference, pages 1–6, 2018. (Cited on page 125.)

[WLS+22] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Etsformer: Exponential smoothing transformers for time-series forecasting. arXiv preprint arXiv:2202.01381, 2022. (Cited on page 53.)

[WSS+05] Shimon Whiteson, Peter Stone, Kenneth O Stanley, Risto Miikkulainen, and Nate Kohl. Automatic feature selection in neuroevolution. In Proceedings of the 7th annual conference on Genetic and evolutionary computation, pages 1225–1232, 2005. (Cited on page 71.)

[WTNM17] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. arXiv preprint arXiv:1711.11053, 2017. (Cited on page 53.)

[WWW+16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In Proceedings of the 30th International Conference on Neural

Information Processing Systems, NIPS'16, page 2082–2090, Red Hook, NY, USA, 2016. Curran Associates Inc. (Cited on page 20.)

[WXWL21] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. Advances in Neural Information Processing Systems, 34:22419–22430, 2021. (Cited on pages 48, 50, 53, and 60.)

[WXZ+22] Zhiyuan Wang, Xovee Xu, Weifeng Zhang, Goce Trajcevski, Ting Zhong, and Fan Zhou. Learning latent seasonal-trend representations for time series forecasting. In Advances in Neural Information Processing Systems, 2022. (Cited on page 53.)

[WZG19] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In International Conference on Learning Representations, 2019. (Cited on pages 5 and 20.)

[WZZ+22] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. arXiv preprint arXiv:2202.07125, 2022. (Cited on page 48.)

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. (Cited on page 30.)

[XWZ+22] Qiao Xiao, Boqian Wu, Yu Zhang, Shiwei Liu, Mykola Pechenizkiy, Elena Mocanu, and Decebal Constantin Mocanu. Dynamic sparse network for time series classification: Learning what to see". In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. (Cited on page 52.)

[XZL15] Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. Information Sciences, 307:39–52, 2015. (Cited on pages 21 and 22.)

[YJS+14] Makoto Yamada, Wittawat Jitkrittum, Leonid Sigal, Eric P Xing, and Masashi Sugiyama. High-dimensional feature selection by

feature-wise kernelized lasso. Neural computation, 26(1):185–207, 2014. (Cited on page 214.)

[YLNK20] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In International Conference on Machine Learning, pages 10648–10659. PMLR, 2020. (Cited on pages 8, 96, 98, and 107.)

[YMN+21] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. Advances in Neural Information Processing Systems, 34:20838–20850, 2021. (Cited on pages 29 and 51.)

[YSM+11] Yi Yang, Heng Tao Shen, Zhigang Ma, Zi Huang, and Xiaofang Zhou. L2, 1-norm regularized discriminative feature selection for unsupervised. In Twenty-Second International Joint Conference on Artificial Intelligence, 2011. (Cited on page 71.)

[YXJ+18] Jun Yang, Wenjing Xiao, Chun Jiang, M Shamim Hossain, Ghulam Muhammad, and Syed Umar Amin. Ai-powered green cloud and data center. IEEE Access, 7:4195–4203, 2018. (Cited on pages 16 and 68.)

[ZCZX22] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? arXiv preprint arXiv:2205.13504, 2022. (Cited on pages xxx, 53, 56, 63, 182, and 184.)

[ZG17] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878, 2017. (Cited on pages xxii, 4, 5, 20, 49, 51, 56, 63, and 177.)

[ZJ19] Hangyu Zhu and Yaochu Jin. Multi-objective evolutionary federated learning. IEEE transactions on neural networks and learning systems, 2019. (Cited on page 74.)

[ZJG+22] Xiyuan Zhang, Xiaoyong Jin, Karthick Gopalswamy, Gaurav Gupta, Youngsuk Park, Xingjian Shi, Hao Wang, Danielle C Maddix, and Bernie Wang. First de-trend then attend: Rethinking

attention for time-series forecasting. In NeurIPS'22 Workshop on All Things Attention: Bridging Different Perspectives on Attention, 2022. (Cited on page 53.)

[ZL07] Zheng Zhao and Huan Liu. Semi-supervised feature selection via spectral analysis. In Proceedings of the 2007 SIAM international conference on data mining, pages 641–646. SIAM, 2007. (Cited on page 68.)

[ZLL+22] Yuxin Zhang, Mingbao Lin, ZhiHang Lin, Yiting Luo, Ke Li, Fei Chao, YONGJIAN WU, and Rongrong Ji. Learning best combination for efficient n:m sparsity. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. (Cited on page 125.)

[ZMW+22] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. arXiv preprint arXiv:2201.12740, 2022. (Cited on pages 48, 50, 53, and 60.)

[ZNLW19] Rui Zhang, Feiping Nie, Xuelong Li, and Xian Wei. Feature selection with multi-view data: A survey. Information Fusion, 50:158–167, 2019. (Cited on pages 7, 96, and 98.)

[ZY23] Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In The Eleventh International Conference on Learning Representations, 2023. (Cited on page 53.)

[ZZC+22] Tianping Zhang, Yizhuo Zhang, Wei Cao, Jiang Bian, Xiaohan Yi, Shun Zheng, and Jian Li. Less is more: Fast multivariate time series forecasting with light sampling-oriented mlp structures. arXiv preprint arXiv:2207.01186, 2022. (Cited on page 53.)

[ZZL+20] Mi Zhang, Faen Zhang, Nicholas D Lane, Yuanchao Shu, Xiao Zeng, Biyi Fang, Shen Yan, and Hui Xu. Deep learning in the era of edge computing: Challenges and opportunities. Fog Computing: Theory and Practice, 2020. (Cited on page 16.)

[ZZP+21] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 11106–11115, 2021. (Cited on pages 48, 50, 53, 60, and 175.)

Appendices

Appendix A

Additional Experiments and Analysis for Chapter 2

A.1 Performance Evaluation

In this Appendix, we conduct a comprehensive performance evaluation of the algorithms discussed in Chapter 2. Particularly, we analyze the results obtained in Section 2.4.2 in more depth. Our analysis encompasses key dimensions such as accuracy, learning speed, and computational complexity. We present learning curves that showcase how these algorithms evolve in terms of classification accuracy across multiple datasets. We introduce a novel metric, Training Delay (TD), to assess learning speed, enabling us to quantify the trade-off between accuracy and efficiency. Furthermore, we delve into the computational complexity of the algorithms during training, comparing CTRE with similar methods and exploring strategies to reduce computational costs. This section offers valuable insights into the overall performance of these algorithms in different scenarios.

A.1.1 Learning Curves

In this section, we include the learning curves for the experiments of Section 2.4.2 in Figures A.1, A.2, A.3, A.4, A.5, and A.6, which corresponds to Madelon, Isolet, MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100, respectively. The characteristics of these datasets are presented in Table 2.1 in Chapter 2.

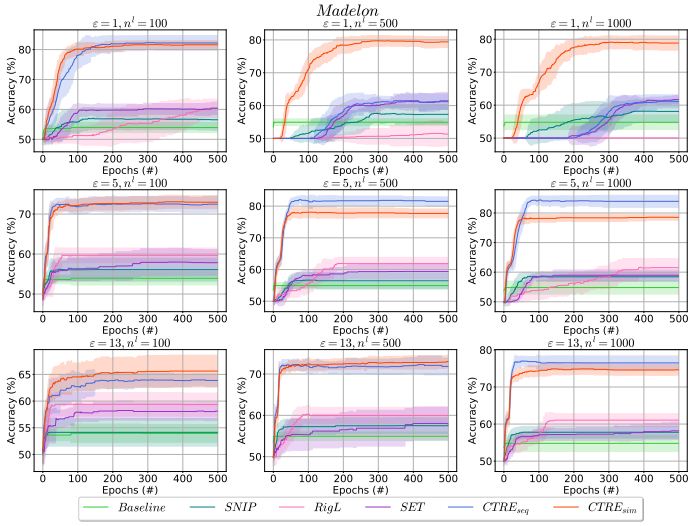


Figure A.1: Classification accuracy (%) results on Madelon.

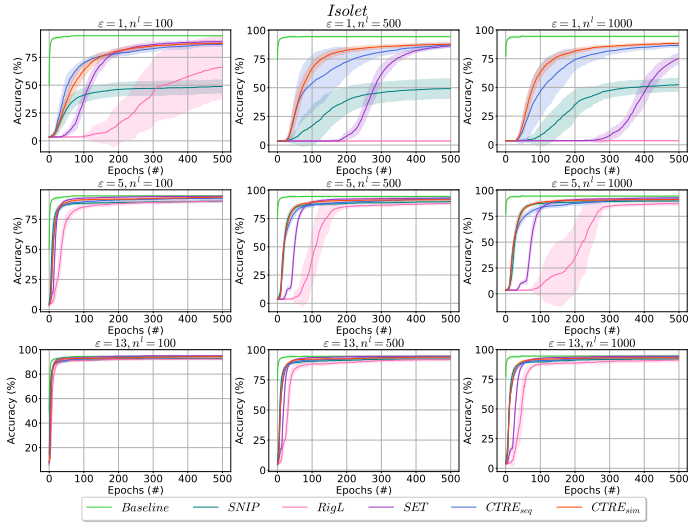


Figure A.2: Classification accuracy (%) results on Isolet.

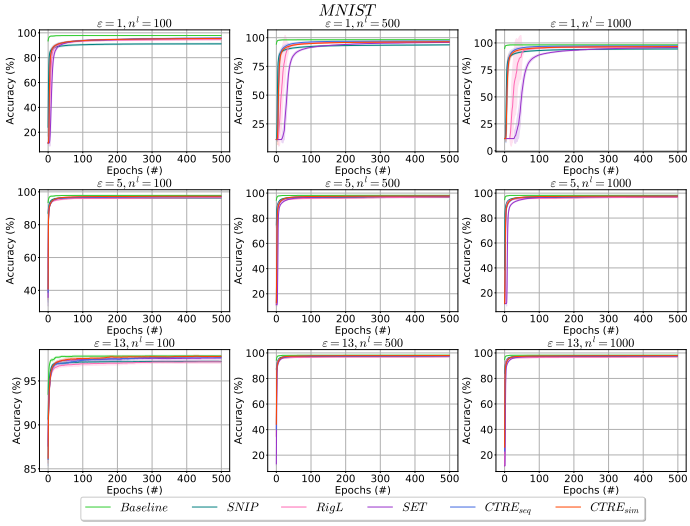


Figure A.3: Classification accuracy (%) results on MNIST.

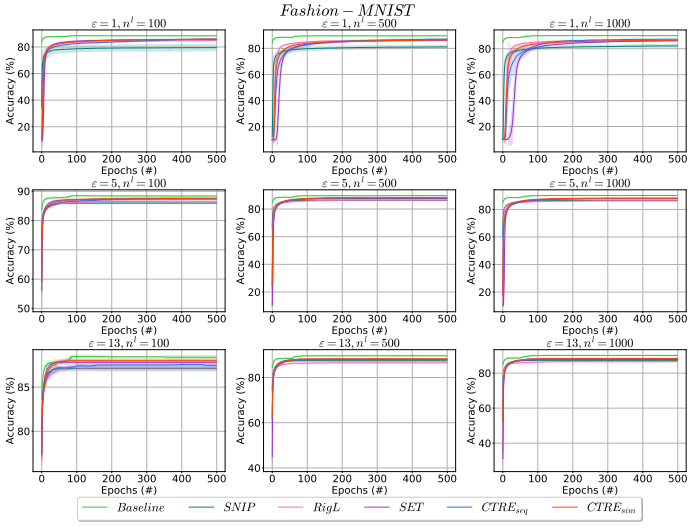


Figure A.4: Classification accuracy (%) results on Fashion-MNIST.

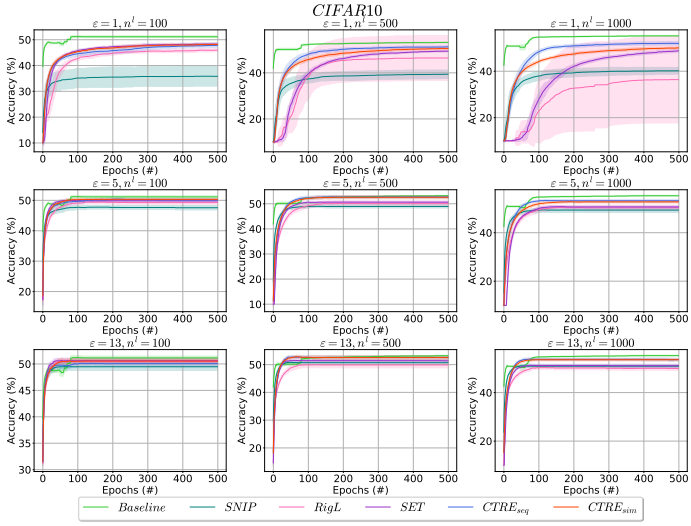


Figure A.5: Classification accuracy (%) results on CIFAR10.

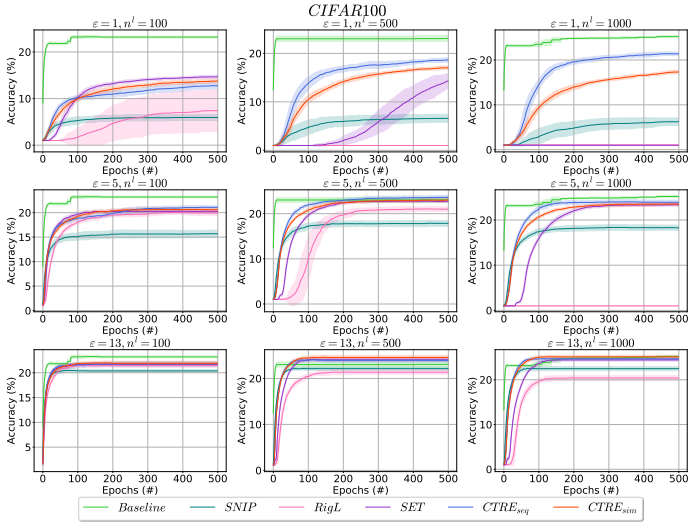


Figure A.6: Classification accuracy (%) results on CIFAR100.

A.1.2 Learning Speed

To compare the training speed, we define a metric that computes the fraction of the total training time required to reach a certain level of accuracy. We call this metric Training Delay (*TD*) and compute it as follows:

$$TD = \frac{\min_{acc_i \geq th \times acc_{max},\, i \in \{1,2,\ldots,\#\,epochs\}} i}{\#\,epochs}, \tag{A.1}$$

where acc_i is the test accuracy at epoch i, th is the threshold hyperparameter between 0 and 1, acc_{max} is the maximum accuracy achieved by the training methods for the model with n^l hidden neurons and sparsity level ε, and $\#\,epochs$ is the total number of training epochs. In other words, *TD* shows the trade-off between accuracy and learning speed. The lower *TD* is for a method, the faster it can be trained to reach a certain desired level of accuracy (determined by th). Therefore, it gains a better trade-off between accuracy and learning speed. We believe that minimizing this metric is crucial for low-resource devices where accuracy is not the only important aspect for evaluating the method's performance. Instead, achieving a decent level of accuracy within a minimum number of training epochs is the primary concern.

For each network with different sizes and training methods, we measure *TD* on all datasets. We consider only the high sparsity case (when $\varepsilon = 1$) since when the network is dense, all the methods have very low *TD*, and the difference between them is negligible. However, when we are looking for a highly sparse sub-network, it takes longer for each method to find the well-performing sub-network, and the difference among the methods is more apparent. We set the

Table A.1: Training delay (TD) (%) comparison among methods. Empty fields indicate that the method cannot reach the considered level of accuracy in 500 training epochs.

		Madelon			**Isolet**			**MNIST**			**Fashion-MNIST**			**CIFAR10**			**CIFAR100**		
		n^l			n^l			n^l			n^l			n^l			n^l		
ε	**Method**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**
	SNIP	–	–	–	–	–	–	**1.8**	2.8	3.2	5.6	8.8	8.6	–	–	–	–	–	–
	RigL	–	–	–	–	–	–	3.2	5.8	9.6	**2.8**	**3.6**	**4.6**	25.2	65.8	–	–	–	–
1	SET	–	–	–	40.0	74.0	–	4.4	11.8	18.2	3.4	9.4	14.2	**11.4**	39.4	68.4	**40.2**	–	–
	$CTRE_{seq}$	16.0	–	–	48.0	52.8	52.4	2.2	**2.6**	**3.0**	3.6	9.2	12.8	13.4	**14.6**	**16.8**	–	**40.4**	**36.6**
	$CTRE_{sim}$	**9.8**	**20.6**	**24.8**	**38.6**	**30.6**	**35.6**	2.2	3.0	3.2	3.0	6.2	7.2	**11.4**	22.4	33.2	66.8	88.6	–

threshold th to 0.9. Therefore, we compute the training delay for reaching 0.9 of the maximum accuracy achieved on this model. The results are presented in Table A.1. If a method cannot reach the $th \times acc_{max}$ within the total number of epochs (500 in these experiments), we keep the corresponding entry empty.

As can be seen in Table A.1, CTRE (including CTRE_{sim} and CTRE_{seq}) has the lowest training delay (*TD*) in 13 out of 18 cases considered. On the Isolet and the Madelon datasets, some methods cannot reach the required level of accuracy (0.9 of the maximum accuracy) within the 500 training epochs. SNIP has the worst performance among these methods and cannot reach the required level of accuracy on Madelon, Isolet, and CIFAR10. RigL has a similar performance to SNIP; while it has a decent performance on Fashion-MNIST and MNIST, it has a poor performance on the other datasets. Finally, SET has comparable performance to other methods on Fashion-MNIST and MNIST. However, when the network is highly sparse and large ($\varepsilon = 1$ and $n^l > 100$) on the Isolet dataset, it does not have a good performance.

A.1.3 Computational Complexity

In this appendix, we compare the algorithms in terms of the computational complexity. While the computational cost during inference is equal for all methods (in the case of having the same sparsity level), the computational complexity during training is different.

We compare the computational complexity with the two closest sparse training algorithms to CTRE: SET and RigL. Our proposed methods require an extra cost of computing the cosine similarity matrix for the connections compared to SET. For each layer in each epoch, CTRE requires computing three dot products of size m (number of samples) for each connection in this layer to compute the similarity matrix in Equation 2.2. Therefore, for each layer l, CTRE requires in the order of $\mathcal{O}(mN^l)$ extra computations at each epoch, where N^l is the number of parameters of layer l. However, this additional cost considerably improves the accuracy and the learning speed (discussed in Appendix A.1.2), particularly on tabular datasets and highly sparse neural networks. Therefore, depending on the application, the specialists should choose the trade-off between accuracy and the computational cost when finding highly sparse neural networks. Compared to RigL, which requires computing occasional dense gradients, CTRE has the same order of complexity. This is because the order of computing gradients for back-propagation is also $\mathcal{O}(mN^l)$. However, CTRE outperforms RigL, especially in the high-sparsity region.

Table A.2: Classification accuracy (%) of $CTRE_{sim}$ with different number of training samples for computing the similarity matrices.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
			ϵ			ϵ			ϵ	
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	$CTRE_{sim}$	81.6 ± 1.3	**73.0 ± 1.6**	**65.6 ± 3.0**	79.4 ± 1.7	77.7 ± 1.4	**73.0 ± 1.5**	78.8 ± 2.2	78.5 ± 1.0	**74.6 ± 1.4**
	$CTRE_{sample2}$	81.0 ± 2.3	72.4 ± 1.4	65.3 ± 1.6	**79.7 ± 0.5**	**77.9 ± 0.9**	71.7 ± 2.8	**80.6 ± 0.7**	78.6 ± 1.0	74.1 ± 0.4
	$CTRE_{sample4}$	**81.9 ± 1.0**	71.3 ± 1.8	64.4 ± 1.0	79.0 ± 1.1	77.8 ± 0.8	72.7 ± 1.6	78.7 ± 1.1	**79.1 ± 1.4**	74.1 ± 0.7
Isolet	$CTRE_{sim}$	**87.5 ± 0.8**	**93.4 ± 0.7**	94.3 ± 0.8	87.8 ± 1.1	91.7 ± 1.1	93.9 ± 0.5	88.3 ± 0.7	91.3 ± 1.3	93.1 ± 0.6
	$CTRE_{sample2}$	81.9 ± 4.2	92.8 ± 0.3	94.7 ± 0.3	**88.4 ± 0.8**	**92.6 ± 0.4**	**94.0 ± 1.1**	88.7 ± 0.5	92.4 ± 0.7	**94.0 ± 0.2**
	$CTRE_{sample4}$	85.0 ± 1.3	93.0 ± 0.8	**94.9 ± 0.5**	87.9 ± 0.5	91.7 ± 1.1	93.8 ± 0.1	**89.4 ± 1.0**	**92.5 ± 0.2**	93.8 ± 0.5
MNIST	$CTRE_{sim}$	95.5 ± 0.2	**97.3 ± 0.1**	**97.8 ± 0.1**	**96.4 ± 0.2**	97.7 ± 0.1	**98.0 ± 0.1**	**96.6 ± 0.2**	97.7 ± 0.1	**97.9 ± 0.1**
	$CTRE_{sample2}$	**95.7 ± 0.0**	97.0 ± 0.1	97.7 ± 0.0	96.3 ± 0.0	**97.8 ± 0.0**	97.8 ± 0.0	96.2 ± 0.0	**97.9 ± 0.1**	97.8 ± 0.1
	$CTRE_{sample4}$	95.1 ± 0.2	**97.3 ± 0.1**	97.6 ± 0.2	96.1 ± 0.2	97.7 ± 0.1	97.8 ± 0.1	**96.6 ± 0.1**	97.7 ± 0.1	97.8 ± 0.0
Fashion-MNIST	$CTRE_{sim}$	**85.8 ± 0.3**	87.5 ± 0.3	**88.0 ± 0.2**	86.4 ± 0.5	**88.1 ± 0.2**	88.3 ± 0.2	86.5 ± 0.3	88.1 ± 0.3	88.3 ± 0.2
	$CTRE_{sample2}$	85.7 ± 0.1	87.4 ± 0.1	87.7 ± 0.3	86.3 ± 0.7	87.9 ± 0.1	88.1 ± 0.2	86.3 ± 0.2	**88.3 ± 0.0**	88.0 ± 0.2
	$CTRE_{sample4}$	85.6 ± 0.4	**87.6 ± 0.1**	**88.0 ± 0.1**	**87.2 ± 0.1**	87.5 ± 0.1	**88.6 ± 0.1**	**86.6 ± 0.3**	88.0 ± 0.1	**88.4 ± 0.2**
CIFAR10	$CTRE_{sim}$	**48.2 ± 0.4**	50.3 ± 0.3	50.5 ± 0.4	50.6 ± 0.4	52.6 ± 0.7	52.5 ± 0.7	50.0 ± 0.4	52.7 ± 0.5	53.5 ± 0.5
	$CTRE_{sample2}$	47.8 ± 0.1	49.4 ± 0.4	50.9 ± 0.4	50.9 ± 0.3	**52.8 ± 0.4**	**52.9 ± 0.4**	50.4 ± 0.5	**53.9 ± 0.1**	53.4 ± 0.2
	$CTRE_{sample4}$	47.8 ± 0.1	**50.8 ± 0.2**	**51.1 ± 0.4**	**52.0 ± 0.1**	52.7 ± 0.3	**52.9 ± 0.6**	**51.6 ± 0.5**	53.6 ± 0.3	**53.6 ± 0.5**

To further decrease the computational cost of CTRE, we have tried to reduce the cost of cosine similarity computation by considering a proportion of the samples to compute the similarity matrix. We run $CTRE_{sim}$ with half of the samples ($CTRE_{sample2}$) and a quarter of samples ($CTRE_{sample4}$) to compute the cosine similarity matrix. The results can be observed in Table A.2. It is clear that even with half of the samples, CTRE can still achieve a close performance as the original method on all datasets. Based on these observations, it can be concluded that only a fraction of samples can be used to compute the similarity matrix. In this way, we would be able to decrease the computational cost without affecting the performance.

Further studies can be performed to decrease the computational cost of deriving the similarity matrix. In short, CTRE is the first step in finding highly sparse neural networks using neuron characteristics and can be further explored in future works.

A.2 Performance Evaluation Using Pure Sparse Implementation

In this appendix, we present the results using the pure sparse implementation. This code is developed from the sparse implementation of SET[1]. While the other training methods for obtaining sparse neural networks mostly use a binary mask over weights to simulate sparsity, this code is implemented in a purely sparse manner using Cython and SciPy sparse matrices. We have implemented our proposed method using this sparse implementation and repeated the experiments from Section 2.4.2. The results are summarized in Table A.3.

Table A.3: Classification accuracy (%) comparison using pure sparse implementation.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
		ϵ			ϵ			ϵ		
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	SET	58.7 ± 2.0	60.4 ± 3.3	58.1 ± 1.5	65.1 ± 2.2	59.6 ± 5.1	61.8 ± 1.0	61.8 ± 1.9	62.1 ± 4.7	61.6 ± 3.9
	$CTRE_{seq}$	85.3 ± 0.3	75.1 ± 0.6	67.1 ± 1.4	**87.2 ± 1.2**	82.1 ± 2.0	75.3 ± 1.0	87.2 ± 0.2	**86.6 ± 0.7**	75.7 ± 2.7
	$CTRE_{sim}$	82.9 ± 1.0	73.9 ± 2.3	66.8 ± 2.7	82.5 ± 1.1	79.4 ± 1.0	74.9 ± 1.2	82.9 ± 1.1	80.4 ± 1.6	74.6 ± 0.6
	$CTRE_{w/oRandom}$	**86.4 ± 1.2**	**75.6 ± 1.2**	**68.5 ± 0.6**	**87.2 ± 1.2**	**82.8 ± 1.1**	**77.0 ± 0.6**	**88.2 ± 0.6**	84.5 ± 0.6	**77.9 ± 1.4**
Isolet	SET	**87.6 ± 1.0**	**94.1 ± 0.1**	94.5 ± 0.4	86.2 ± 1.2	93.7 ± 0.7	94.4 ± 0.1	86.1 ± 0.5	94.1 ± 0.1	94.7 ± 0.1
	$CTRE_{seq}$	83.1 ± 1.2	93.7 ± 0.4	**94.6 ± 0.7**	**91.1 ± 0.3**	**94.2 ± 0.3**	**94.5 ± 0.4**	**90.9 ± 1.1**	**94.3 ± 0.4**	**94.8 ± 0.3**
	$CTRE_{sim}$	85.6 ± 0.6	93.2 ± 0.7	**94.6 ± 0.6**	88.6 ± 1.3	93.7 ± 0.4	94.4 ± 0.2	88.8 ± 0.6	93.5 ± 0.1	94.0 ± 0.1
	$CTRE_{w/oRandom}$	81.1 ± 1.1	92.9 ± 1.6	**94.6 ± 0.2**	89.1 ± 0.5	93.8 ± 0.1	93.9 ± 0.3	90.6 ± 0.9	93.9 ± 0.4	93.4 ± 0.7
MNIST	SET	95.0 ± 0.2	97.5 ± 0.1	97.8 ± 0.1	94.9 ± 0.2	97.5 ± 0.0	97.8 ± 0.1	95.2 ± 0.2	97.3 ± 0.1	97.6 ± 0.0
	$CTRE_{seq}$	**95.7 ± 0.2**	**97.8 ± 0.1**	**97.9 ± 0.0**	97.3 ± 0.2	**97.9 ± 0.2**	**98.0 ± 0.0**	**97.6 ± 0.0**	**97.9 ± 0.1**	**97.9 ± 0.1**
	$CTRE_{sim}$	95.4 ± 0.1	97.6 ± 0.0	97.7 ± 0.0	96.4 ± 0.6	97.4 ± 0.1	97.8 ± 0.0	96.7 ± 0.6	97.2 ± 0.0	97.7 ± 0.1
	$CTRE_{w/oRandom}$	95.5 ± 0.1	97.4 ± 0.0	97.7 ± 0.1	**97.4 ± 0.1**	97.7 ± 0.1	97.5 ± 0.2	**97.6 ± 0.1**	97.8 ± 0.0	97.8 ± 0.2
Fashion-MNIST	SET	**85.6 ± 0.1**	87.7 ± 0.1	**88.5 ± 0.1**	85.2 ± 0.2	87.9 ± 0.1	88.4 ± 0.3	85.2 ± 0.2	87.8 ± 0.0	88.5 ± 0.2
	$CTRE_{seq}$	85.4 ± 0.4	**88.0 ± 0.1**	88.4 ± 0.0	**86.8 ± 0.1**	**88.5 ± 0.2**	**88.6 ± 0.1**	**87.1 ± 0.4**	**88.6 ± 0.2**	**88.7 ± 0.2**
	$CTRE_{sim}$	84.9 ± 0.8	87.4 ± 0.3	88.1 ± 0.1	85.7 ± 0.6	87.7 ± 0.1	88.2 ± 0.1	85.9 ± 0.7	87.5 ± 0.1	88.4 ± 0.2
	$CTRE_{w/oRandom}$	83.7 ± 0.2	87.3 ± 0.2	87.7 ± 0.2	85.9 ± 0.2	87.9 ± 0.3	88.0 ± 0.1	86.5 ± 0.4	88.0 ± 0.2	87.9 ± 0.0
CIFAR10	SET	48.5 ± 0.4	52.7 ± 0.5	**54.0 ± 0.4**	47.7 ± 0.4	53.3 ± 0.3	54.3 ± 0.4	46.2 ± 0.2	52.8 ± 0.2	54.5 ± 0.7
	$CTRE_{seq}$	48.5 ± 0.5	**53.3 ± 0.3**	53.0 ± 0.6	**52.2 ± 0.3**	**55.7 ± 0.6**	**55.6 ± 0.3**	**54.2 ± 0.3**	**55.8 ± 0.1**	**56.2 ± 0.1**
	$CTRE_{sim}$	**48.6 ± 0.0**	51.0 ± 0.0	51.3 ± 0.0	50.1 ± 0.6	55.4 ± 0.1	54.3 ± 0.2	50.2 ± 0.9	**55.8 ± 0.4**	55.8 ± 0.0
	$CTRE_{w/oRandom}$	45.5 ± 0.5	49.6 ± 0.3	49.0 ± 0.4	51.2 ± 0.3	53.5 ± 0.3	53.0 ± 0.1	53.8 ± 0.4	53.6 ± 0.0	54.7 ± 0.1

As can be seen in Table A.3, the results are subtly different from Table 2.2 in Section 2.4.2. This difference arises from some small differences in the implementation. One of the main differences is that in Section 2.4.2, the TensorFlow library is used for implementing the neural network. However,

[1]The pure sparse implementation of SET can be found on *https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks*.

this implementation uses Numpy, Scipy, and Cython to perform sparse matrix operations. Another difference is the weight initialization policy. While in the experiments of Section 2.4.2, weights are initialized using a uniform distribution, in the sparse implementation weights are initialized using a normal distribution which seems more beneficial to this implementation. Overall, in most cases, the results in Table A.3 are higher than the results in Table 2.2.

A.3 Cosine vs. Euclidean-based Similarity Metric

In this section, we analyze the effectiveness of the cosine similarity metric in evolving the topology of the sparse neural network compared to other similarity metrics. To achieve this, we consider Euclidean-based similarity metric. Instead of computing the importance of the non-existing connections using cosine similarity (Equation 2.2), we compute it as:

$$Sim^{l}_{p,q} = \frac{1}{1 + d(\boldsymbol{A}^{l}_{:,q}, \boldsymbol{A}^{l-1}_{:,p})}, \tag{A.2}$$

where $d(a,b)$ is the Euclidean distance between vectors a and b. We replace Equation A.2 with Equation 2.2 in Algorithm 2 and we call this method as $\text{CTRE}_{\text{sim-euclidean}}$. We compare $\text{CTRE}_{\text{sim-euclidean}}$ with CTRE_{sim}. The results are presented in Table A.4.

As shown in Table A.4, CTRE_{sim} outperforms $\text{CTRE}_{\text{sim-euclidean}}$ in most cases considered. Particularly, in the high sparsity regime ($\varepsilon = 1$), there is a considerable gap between the performance of these two methods. It can be concluded that the Euclidean-based similarity metric is not very informative in evolving the topology of sparse neural networks. This might stem from the sensitivity of this metric to the vectors' magnitude. Cosine-similarity metric, a magnitude-insensitive metric that also presents a biologically plausible approach for measuring the importance of the connections, is a good choice for obtaining sparse neural networks in the CTRE algorithm.

A.4 Ablation Study: Cosine and Random-based Weight Addition Order in CTRE_{seq}

In this section, we perform an ablation study on the CTRE_{seq} algorithm to measure the effectiveness of the cosine-based and random search order in the

Table A.4: Classification accuracy (%) comparison among cosine and euclidean-based similarity metrics in CTRE algorithm.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
			ϵ			ϵ			ϵ	
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	CTRE_{sim}	**81.6±1.3**	**73.0±1.6**	**65.6±3.0**	**79.4±1.7**	**77.7±1.4**	**73.0±1.5**	**78.8±2.2**	**78.5±1.0**	**74.6±1.4**
	$\text{CTRE}_{\text{sim-euclidean}}$	77.2±1.7	65.6±1.3	63.7±0.7	71.5±4.2	74.2±1.6	65.8±4.3	70.6±4.5	76.9±1.8	67.8±2.2
Isolet	CTRE_{sim}	**87.5±0.8**	93.4±0.7	94.3±0.8	**87.8±1.1**	**91.7±1.1**	**93.9±0.5**	**88.3±0.7**	91.3±1.3	93.1±0.6
	$\text{CTRE}_{\text{sim-euclidean}}$	83.0±1.3	**93.7±0.6**	**94.5±0.7**	72.0±4.1	91.6±0.3	93.7±0.8	49.8±6.1	**91.4±0.7**	**93.6±0.5**
MNIST	CTRE_{sim}	**95.5±0.2**	**97.3±0.1**	**97.8±0.1**	**96.4±0.2**	**97.7±0.1**	**98.0±0.1**	**96.6±0.2**	**97.7±0.1**	**97.9±0.1**
	$\text{CTRE}_{\text{sim-euclidean}}$	94.6±0.2	97.0±0.1	97.3±0.1	94.3±0.2	97.2±0.1	97.4±0.0	93.8±0.1	96.9±0.0	97.1±0.3
Fashion-MNIST	CTRE_{sim}	**85.8±0.3**	**87.5±0.3**	**88.0±0.2**	**86.4±0.5**	**88.1±0.2**	**88.3±0.2**	**86.5±0.3**	**88.1±0.3**	**88.3±0.2**
	$\text{CTRE}_{\text{sim-euclidean}}$	84.9±0.2	87.1±0.1	87.7±0.1	84.6±0.2	87.4±0.2	87.8±0.2	83.8±0.3	87.4±0.1	87.4±0.4
CIFAR10	CTRE_{sim}	**48.2±0.4**	**50.3±0.3**	50.5±0.4	**50.6±0.4**	**52.6±0.7**	**52.5±0.7**	**50.0±0.4**	**52.7±0.5**	**53.5±0.5**
	$\text{CTRE}_{\text{sim-euclidean}}$	46.1±0.4	50.1±0.4	**50.7±0.4**	46.1±1.4	50.5±0.5	51.6±0.2	44.8±0.5	49.5±0.6	51.2±0.3
CIFAR100	CTRE_{sim}	**13.8±0.4**	20.6±0.4	**21.9±0.4**	**17.0±0.3**	**23.0±0.3**	**24.6±0.4**	**17.3±0.4**	**23.5±0.3**	**25.1±0.3**
	$\text{CTRE}_{\text{sim-euclidean}}$	12.2±0.4	**20.7±0.4**	21.8±0.3	11.8±0.7	22.5±0.3	24.0±0.2	9.4±1.0	21.9±0.3	23.8±0.4

performance of the algorithm. With this aim, we measure the performance of two variants of CTRE_{seq} (Algorithm 1) that are different from CTRE_{seq} in the order of the cosine and random weight addition:

- **$\text{CTRE}_{\text{abl1:seq}}$** starts training with random weight addition, and switches to cosine-based addition when there is no improvement in the validation accuracy for $e_{early\,stop}$ epochs.

- **$\text{CTRE}_{\text{abl2:seq}}$** splits the training process into two equal phases in terms of the number of epochs. In the first phase, it adds weights randomly, and in the second phase, it uses the cosine-similarity information for weight addition.

We compare the performance of these two algorithms with CTRE_{seq} in Table A.5. CTRE_{seq} outperforms $\text{CTRE}_{\text{abl1:seq}}$ and $\text{CTRE}_{\text{abl2:seq}}$ in the majority of the experiments. In high sparsity regime and large network size ($\epsilon = 1$, $n^l = 1000$), there is a considerable gap between their performance. The reason behind this difference is that by using cosine similarity-based weight addition at the beginning of the algorithm, the algorithm finds a well-performing sub-network very fast. Then, in the rest of the algorithm, this topology is improved using cosine information and random search. However, by starting with random weight addition, it might take longer for the algorithm to reach a reasonable

Table A.5: Classification accuracy (%) comparison among variants of CTRE_{seq} algorithm.

		$n^l = 100$			$n^l = 500$			$n^l = 1000$		
			ϵ			ϵ			ϵ	
Dataset	**Method**	**1**	**5**	**13**	**1**	**5**	**13**	**1**	**5**	**13**
Madelon	CTRE_{seq}	**82.2 ± 2.4**	**72.5 ± 2.0**	**63.9 ± 1.2**	**61.2 ± 2.4**	**81.5 ± 1.4**	**71.8 ± 2.0**	61.1 ± 1.9	**83.9 ± 2.0**	**76.5 ± 1.9**
	$\text{CTRE}_{\text{abl1:seq}}$	57.8 ± 2.3	58.1 ± 3.1	57.8 ± 2.1	61.0 ± 3.1	59.6 ± 3.0	59.4 ± 2.0	61.1 ± 2.9	59.7 ± 1.1	58.3 ± 2.4
	$\text{CTRE}_{\text{abl2:seq}}$	59.2 ± 3.0	58.0 ± 3.5	58.4 ± 2.3	61.1 ± 2.0	58.7 ± 2.9	58.8 ± 2.4	**61.6 ± 0.9**	58.4 ± 1.6	58.7 ± 1.9
Isolet	CTRE_{seq}	86.7 ± 1.8	92.4 ± 1.1	94.3 ± 0.5	**87.2 ± 2.0**	92.3 ± 0.6	94.0 ± 0.4	**86.7 ± 1.6**	91.5 ± 1.0	**93.7 ± 0.5**
	$\text{CTRE}_{\text{abl1:seq}}$	**89.5 ± 1.1**	93.7 ± 0.5	**94.9 ± 0.5**	85.8 ± 1.0	**93.3 ± 0.8**	**94.4 ± 0.4**	72.5 ± 2.3	92.3 ± 0.6	93.5 ± 0.9
	$\text{CTRE}_{\text{abl2:seq}}$	89.0 ± 1.3	**93.8 ± 0.6**	94.6 ± 0.7	85.8 ± 1.5	93.1 ± 0.8	94.2 ± 0.9	75.1 ± 3.9	**92.4 ± 0.9**	93.3 ± 0.7
MNIST	CTRE_{seq}	**95.7 ± 0.2**	**97.3 ± 0.2**	**97.7 ± 0.1**	**97.0 ± 0.2**	**97.6 ± 0.2**	**97.8 ± 0.1**	**97.3 ± 0.1**	**97.7 ± 0.1**	**97.8 ± 0.1**
	$\text{CTRE}_{\text{abl1:seq}}$	95.5 ± 0.3	97.1 ± 0.1	97.6 ± 0.1	95.8 ± 0.2	97.3 ± 0.1	97.5 ± 0.1	95.9 ± 0.1	97.3 ± 0.1	97.2 ± 0.1
	$\text{CTRE}_{\text{abl2:seq}}$	95.5 ± 0.2	97.1 ± 0.1	97.6 ± 0.1	96.1 ± 0.1	97.4 ± 0.2	97.5 ± 0.1	96.0 ± 0.1	97.4 ± 0.1	97.3 ± 0.0
Fashion-MNIST	CTRE_{seq}	85.8 ± 0.5	**87.5 ± 0.3**	87.4 ± 0.3	**87.1 ± 0.4**	**87.9 ± 0.3**	**88.0 ± 0.2**	**87.3 ± 0.2**	**88.0 ± 0.3**	**88.3 ± 0.2**
	$\text{CTRE}_{\text{abl1:seq}}$	85.7 ± 0.2	**87.5 ± 0.2**	87.8 ± 0.2	86.3 ± 0.3	87.7 ± 0.2	87.9 ± 0.3	86.0 ± 0.2	87.7 ± 0.3	87.5 ± 0.3
	$\text{CTRE}_{\text{abl2:seq}}$	**85.9 ± 0.3**	87.4 ± 0.2	**87.9 ± 0.2**	86.4 ± 0.2	87.7 ± 0.0	87.7 ± 0.2	86.2 ± 0.1	87.7 ± 0.1	87.6 ± 0.2
CIFAR10	CTRE_{seq}	47.9 ± 0.8	**50.1 ± 0.5**	50.1 ± 0.5	**51.3 ± 0.5**	**52.7 ± 0.5**	**52.6 ± 0.6**	**52.0 ± 0.7**	**53.2 ± 0.6**	**53.6 ± 0.6**
	$\text{CTRE}_{\text{abl1:seq}}$	**48.1 ± 0.4**	49.3 ± 0.6	**50.5 ± 0.4**	49.3 ± 0.4	50.9 ± 0.4	51.1 ± 0.5	48.9 ± 0.4	50.9 ± 0.4	50.8 ± 0.7
	$\text{CTRE}_{\text{abl2:seq}}$	47.8 ± 0.2	49.9 ± 0.4	50.3 ± 0.2	49.2 ± 0.4	50.8 ± 0.4	51.1 ± 0.2	48.6 ± 0.1	51.0 ± 0.1	50.7 ± 0.3
CIFAR100	CTRE_{seq}	12.7 ± 0.7	**21.1 ± 0.3**	**21.8 ± 0.5**	**18.7 ± 0.4**	**23.6 ± 0.4**	**24.0 ± 0.4**	**21.4 ± 0.4**	**23.9 ± 0.5**	**24.7 ± 0.4**
	$\text{CTRE}_{\text{abl1:seq}}$	**14.9 ± 0.6**	20.5 ± 0.3	21.5 ± 0.5	13.9 ± 2.3	22.8 ± 0.3	23.8 ± 0.6	1.0 ± 0.0	23.6 ± 0.3	24.3 ± 0.4
	$\text{CTRE}_{\text{abl2:seq}}$	14.5 ± 0.3	20.4 ± 0.1	21.4 ± 0.3	14.1 ± 0.9	22.8 ± 0.0	**24.0 ± 0.1**	1.0 ± 0.0	23.4 ± 0.4	24.6 ± 0.2

level of performance. This results in a lower accuracy than CTRE_{seq} at the end of training.

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

Appendix B

Additional Experiments and Analysis for Chapter 3

B.1 Experimental Settings

B.1.1 Datasets

The datasets are summarized in Table 3.2.

1. *Electricity*[1] dataset includes the hourly electricity consumption for 321 consumers between 2012 and 2014.

2. *ETT* [ZZP+21] (Electricity Transformer Temperature) dataset contains load and oil temperature measurements from electricity transformers. 3)

3. *Exchange* [LCYL18] dataset consists of daily exchange rates from 8 countries between 1990 and 2016.

4. *Illness*[2] dataset includes weekly collected data from influenza-like illness patients between 2002 and 2021 reported by Centers for Disease Control and Prevention of the United States.

[1] *https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014*
[2] *https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html*

5. *Traffic*[3] dataset contains road occupancy rates in San Francisco Bay area freeways.

6. *Weather*[4] consists of measurements of 21 weather indicators collected every 10 minutes in 2020.

All datasets are divided in chronological order into the train, validation, and test sets with a split ratio of 7:2:2 (except for the ETT dataset where we use 6:2:2 split ratio).

B.1.2 Prediction Quality Evaluation Metrics

We use MSE and MAE as the evaluation metrics, which can be computed as below:

$$MSE(\widetilde{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H}) = \frac{1}{H}\Sigma_{i=0}^{H-1}(\widetilde{\boldsymbol{x}}_{t+i} - \boldsymbol{x}_{t+i})^2. \tag{B.1}$$

$$MAE(\widetilde{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H}) = \frac{1}{H}\Sigma_{i=0}^{H-1}|\widetilde{\boldsymbol{x}}_{t+i} - \boldsymbol{x}_{t+i}|. \tag{B.2}$$

B.1.3 Hyperparameters

The settings of the transformer models and the hyperparameter values are adopted from the NSTransformer implementation[5]. Sequence length L was set to 36 for the Illness dataset and 96 for the other datasets. Several values for prediction length were tested in the experiments: $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$). The models considered in the experiments are all trained with the ADAM optimizer with a learning rate of 10^{-4}. The batch size used in the experiments was equal to 32. A maximum number of 10 epochs was used for each experiment. Each transformer model consisted of two encoder layers and one decoder layer. Model dimension d_{model} was set to 512 in the experiments unless stated otherwise. PALS starts from a dense model (Initial density $D_{init} = 1$). The pruning rate ζ is initialized to 0.5 and decreased during training with a cosine decay schedule. The values of pruning rate factor γ and loss freedom factor λ are optimized in $\{1.05, 1.1, 1.2\}$ on the validation set using a single random seed for each experiment. Minimum sparsity S_{min} and

[3] *https://pems.dot.ca.gov/*
[4] *https://www.bgc-jena.mpg.de/wetter/*
[5] *https://github.com/thuml/Nonstationary_Transformers*

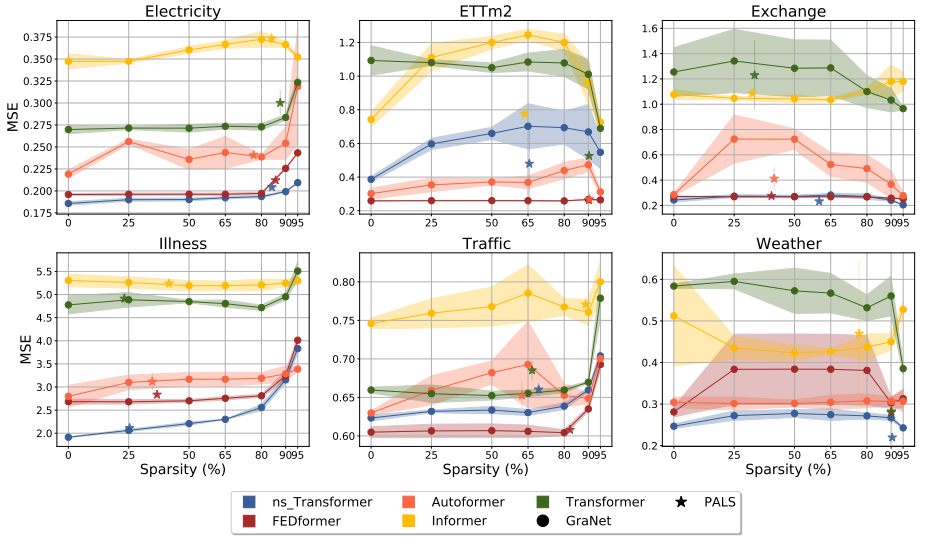
maximum sparsity S_{max} are set to 20 and 90, respectively. These values give the flexibility to the user to tune the range of the model sparsity based on the resource availability in their application. The mask update frequency Δt was equal to 5 for the Illness dataset and 20 for the other datasets. Each experiment was run on three different random seeds and the average measurements are reported for each metric.

B.2 Analyzing Sparsity Effect in Transformers for Time Series Forecasting

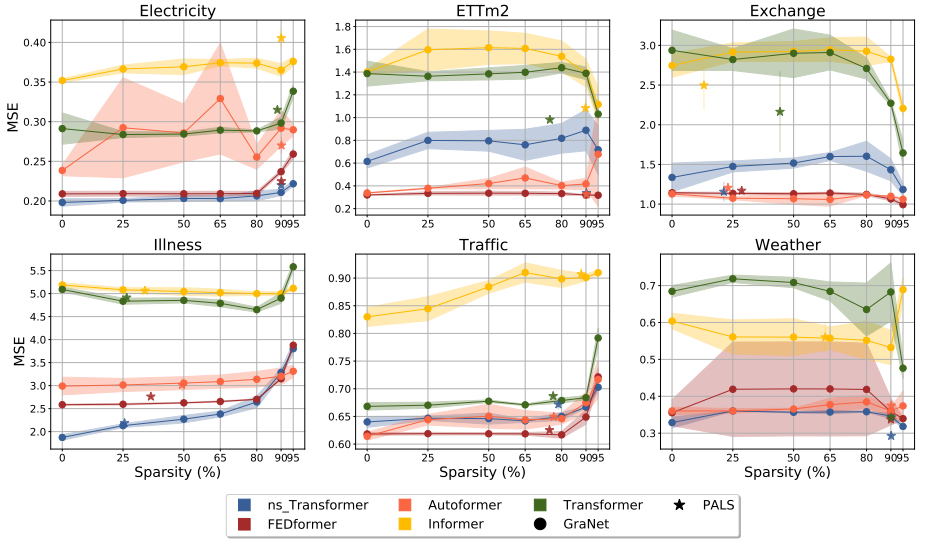
This section presents the results for the sparsity effect in various time series forecasting transformers. Specifically, we employ GraNet [LCC+21] to prune each transformer model, and then evaluate their effectiveness at different sparsity levels.

GraNet gradually prunes a network (in this work, we start from a dense network) during the training to reach a pre-determined sparsity level while allowing for connection regeneration. Therefore, it takes advantage of dense-to-sparse and sparse-to-sparse training by exploring faster the search space and dynamically optimizing the sparse connectivity during training, respectively, to find a decent connectivity pattern efficiently. During training, GraNet executes gradual pruning and zero-cost Neuroregeneration every δt iterations. Gradual pruning gradually reduces network density towards a specific sparsity level across multiple pruning iterations. The initial sparsity level can be zero (creating a dense network, resulting in a dense-to-sparse approach) or higher (starting from a random sparse topology, resulting in a sparse-to-sparse approach). At each pruning step, a portion of weights with the lowest magnitudes is pruned, based on a fixed schedule. This stage is similar to gradual magnitude pruning (GMP) [ZG17]. Following each pruning step, a zero-cost Neuroregeneration is executed. This involves dropping a portion of the existing connections with low magnitudes, which are considered damaged, and adding an equal number of new connections back to the network. The new connections are chosen from non-existing connections with the highest gradient value.

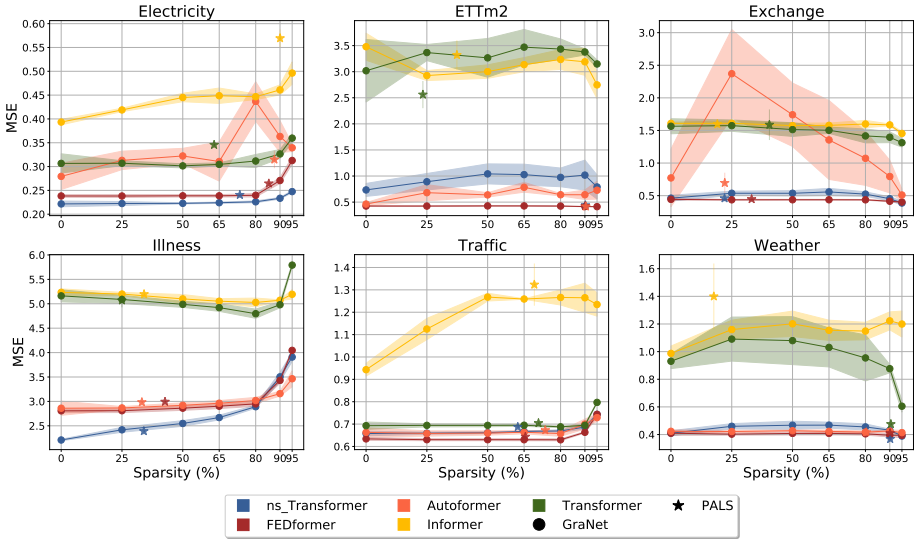
The results for the sparsity effect on the performance of various transformer models are presented in Figure 3.2 in Chapter 3 and Figure B.1. The findings are discussed in depth in Section 3.3 in Chapter 3.



(a) Sparsity effect for $H = 192$ ($H = 36$ for the Illness dataset) in terms of MSE loss.



(b) Sparsity effect for $H = 336$ ($H = 48$ for the Illness dataset) in terms of MSE loss.



(c) Sparsity effect for $H = 720$ ($H = 60$ for the Illness dataset) in terms of MSE loss.

Figure B.1: Sparsity effect on the performance of various transformer models for time series forecasting on benchmark datasets in terms of MSE loss for various prediction lengths as indicated in each figure. Each model is sparsified using GraNet [LCC+21] to sparsity levels (%) $\in \{25, 50, 65, 80, 90, 95\}$. Sparsity=0 indicates the original dense model.

B.3 Comparison Results

The detailed results of the experiments performed in Section 3.5.2 are presented in Table B.1.

B.4 Univariate results

The results are presented in Figures B.2 and B.3 and discussed in Section 3.5.2 in Chapter 3.

Table B.1: Comparison on the benchmark Datasets for prediction length $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$). For each model, the performance of the original (Dense) and the pruned model using PALS is presented in terms of MSE and parameter count ($\times 10^6$). The difference between the results compared to the dense counterpart is shown in parenthesis as % (blue indicates improvement in the results compared to the dense model). **Bold** entries are the best performer in each experiment among various models.

Datasets	H	NSTransformer				FEDformer				Autoformer				Informer				Transformer			
		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS	
		MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param
Electricity	96	**0.170**	12.2	0.195 (14.3% ↑)	2.6 (78.2% ↓)	0.187	19.5	0.203 (8.6% ↑)	4.4 (77.3% ↓)	0.207	12.1	0.206 (0.4% ↓)	5.3 (56.2% ↓)	0.333	12.5	0.347 (4.3% ↑)	1.2 (90.1% ↓)	0.257	11.7	0.282 (9.6% ↑)	3.0 (73.9% ↓)
	192	**0.186**	12.2	0.204 (10.0% ↑)	1.9 (84.3% ↓)	0.196	19.5	0.212 (8.3% ↑)	2.8 (85.9% ↓)	0.219	12.1	0.241 (9.9% ↑)	2.8 (76.7% ↓)	0.347	12.5	0.373 (7.4% ↑)	2.0 (84.2% ↓)	0.270	11.7	0.300 (11.3% ↑)	1.4 (87.8% ↓)
	336	**0.198**	11.9	0.220 (11.1% ↑)	1.2 (90.0% ↓)	0.209	19.5	0.225 (7.7% ↑)	1.9 (90.1% ↓)	0.239	12.1	0.270 (13.2% ↑)	1.2 (90.2% ↓)	0.352	12.5	0.405 (15.2% ↑)	1.2 (90.1% ↓)	0.291	11.7	0.315 (8.2% ↑)	1.3 (88.5% ↓)
	720	**0.222**	11.9	0.241 (8.6% ↑)	3.2 (73.5% ↓)	0.238	19.5	0.265 (11.0% ↑)	2.8 (85.4% ↓)	0.279	12.1	0.315 (12.7% ↑)	1.5 (87.7% ↓)	0.393	12.5	0.569 (44.8% ↑)	1.2 (90.0% ↓)	0.307	11.7	0.346 (12.7% ↑)	4.3 (62.9% ↓)
ETTm2	96	0.241	10.6	0.244 (1.0% ↑)	4.1 (61.0% ↓)	**0.190**	17.9	0.201 (5.7% ↑)	1.8 (90.2% ↓)	0.233	10.5	0.214 (7.9% ↓)	1.0 (90.1% ↓)	0.485	11.3	0.371 (23.3% ↓)	8.7 (22.9% ↓)	0.418	10.5	0.268 (35.8% ↓)	1.0 (90.8% ↓)
	192	0.387	10.7	0.480 (24.1% ↑)	3.7 (65.6% ↓)	**0.259**	17.9	0.263 (1.7% ↑)	1.8 (90.2% ↓)	0.302	10.5	0.269 (10.8% ↓)	1.0 (90.4% ↓)	0.742	11.3	0.777 (4.7% ↑)	4.1 (63.6% ↓)	1.093	10.5	0.527 (51.8% ↓)	1.0 (90.3% ↓)
	336	0.615	10.6	0.339 (44.9% ↓)	1.0 (90.2% ↓)	**0.320**	17.9	0.325 (1.4% ↑)	1.8 (90.2% ↓)	0.338	10.5	0.326 (3.4% ↓)	1.0 (90.2% ↓)	1.400	11.3	1.084 (22.5% ↓)	1.2 (89.8% ↓)	1.387	10.5	0.982 (29.2% ↓)	2.6 (75.2% ↓)
	720	0.734	10.6	0.439 (40.1% ↓)	1.0 (90.2% ↓)	0.425	17.9	0.423 (0.4% ↓)	1.8 (90.1% ↓)	0.463	10.5	**0.417** (9.9% ↓)	1.0 (90.4% ↓)	3.479	11.3	3.320 (4.5% ↓)	7.1 (37.4% ↓)	3.018	10.5	2.561 (15.1% ↓)	8.1 (23.4% ↓)
Exchange	96	0.124	10.5	**0.116** (6.8% ↓)	1.0 (90.4% ↓)	0.146	17.9	0.155 (5.7% ↑)	6.7 (62.7% ↓)	0.147	10.5	0.148 (0.4% ↑)	5.9 (44.2% ↓)	0.941	11.3	0.945 (0.5% ↑)	7.7 (32.1% ↓)	0.679	10.5	0.651 (4.2% ↓)	7.2 (31.6% ↓)
	192	0.243	10.5	**0.233** (4.4% ↓)	4.2 (60.2% ↓)	0.271	17.9	0.276 (1.6% ↑)	10.6 (40.5% ↓)	0.285	10.5	0.412 (44.3% ↑)	6.2 (41.6% ↓)	1.077	11.3	1.087 (1.0% ↑)	7.7 (32.5% ↓)	1.255	10.5	1.229 (2.0% ↓)	7.0 (33.5% ↓)
	336	0.462	10.6	0.459 (0.5% ↓)	8.2 (22.0% ↓)	**0.443**	17.9	0.447 (0.9% ↑)	11.9 (33.2% ↓)	0.772	10.5	0.695 (9.9% ↓)	8.2 (22.1% ↓)	1.609	11.3	1.596 (0.8% ↓)	9.2 (19.1% ↓)	1.565	10.5	1.588 (1.5% ↑)	6.3 (40.5% ↓)
	720	1.336	10.6	1.156 (13.5% ↓)	8.3 (21.3% ↓)	1.143	17.9	1.169 (2.3% ↑)	12.8 (28.6% ↓)	**1.128**	10.5	1.206 (6.9% ↑)	8.1 (23.0% ↓)	2.747	11.3	2.497 (9.1% ↓)	9.8 (13.1% ↓)	2.938	10.5	2.164 (26.3% ↓)	5.9 (44.4% ↓)
Illness	24	**2.555**	10.5	2.641 (3.3% ↑)	6.9 (35.0% ↓)	3.285	13.1	3.590 (9.3% ↑)	7.6 (42.1% ↓)	3.686	10.5	3.648 (1.0% ↓)	7.5 (29.2% ↓)	5.361	11.3	5.412 (1.0% ↑)	7.9 (29.9% ↓)	4.740	10.5	4.739 (0.0% ↓)	6.9 (34.3% ↓)
	36	**1.913**	10.5	2.111 (10.4% ↑)	7.9 (25.4% ↓)	2.682	13.5	2.835 (5.7% ↑)	8.5 (36.7% ↓)	2.799	10.5	3.118 (11.4% ↑)	6.9 (34.8% ↓)	5.304	11.3	5.239 (1.2% ↓)	6.6 (41.7% ↓)	4.776	10.5	4.911 (2.8% ↑)	8.1 (23.2% ↓)
	48	**1.873**	10.5	2.186 (16.7% ↑)	7.8 (25.6% ↓)	2.585	13.9	2.762 (6.9% ↑)	8.8 (36.4% ↓)	2.990	10.5	3.017 (0.9% ↑)	5.3 (49.3% ↓)	5.187	11.3	5.064 (2.4% ↓)	7.5 (34.0% ↓)	5.090	10.5	4.914 (3.5% ↓)	7.7 (26.5% ↓)
	60	**2.209**	10.5	2.391 (8.2% ↑)	7.0 (34.0% ↓)	2.807	14.3	2.994 (6.7% ↑)	8.2 (42.7% ↓)	2.860	10.5	2.984 (4.3% ↑)	7.0 (33.2% ↓)	5.234	11.3	5.197 (0.7% ↓)	7.5 (34.1% ↓)	5.162	10.5	5.070 (1.8% ↓)	7.9 (25.0% ↓)
Traffic	96	0.608	14.1	0.641 (5.4% ↑)	4.3 (69.6% ↓)	**0.581**	22.3	0.585 (0.8% ↑)	5.4 (75.6% ↓)	0.649	14.9	0.629 (3.1% ↓)	5.2 (65.2% ↓)	0.724	14.4	0.744 (2.7% ↑)	1.5 (89.3% ↓)	0.646	13.6	0.677 (4.8% ↑)	3.7 (72.8% ↓)
	192	0.623	14.1	0.660 (5.9% ↑)	4.3 (69.4% ↓)	**0.605**	22.3	0.608 (0.5% ↑)	3.9 (82.4% ↓)	0.630	14.9	0.653 (3.6% ↑)	5.6 (62.4% ↓)	0.746	14.4	0.771 (3.3% ↑)	1.6 (88.9% ↓)	0.660	13.6	0.685 (3.9% ↑)	4.5 (66.7% ↓)
	336	0.640	14.6	0.672 (5.0% ↑)	3.1 (78.7% ↓)	0.618	22.3	0.625 (1.1% ↑)	5.6 (75.0% ↓)	**0.614**	14.9	0.649 (5.7% ↑)	3.5 (76.8% ↓)	0.830	14.4	0.907 (9.3% ↑)	1.7 (88.1% ↓)	0.668	13.6	0.687 (2.8% ↑)	3.2 (76.5% ↓)
	720	0.658	14.1	0.687 (4.5% ↑)	5.3 (62.4% ↓)	**0.634**	22.3	0.643 (1.5% ↑)	7.7 (65.5% ↓)	0.663	14.9	0.673 (1.5% ↑)	3.9 (73.7% ↓)	0.943	14.4	1.324 (40.3% ↑)	4.4 (69.2% ↓)	0.693	13.6	0.705 (1.7% ↑)	4.0 (70.9% ↓)
Weather	96	0.182	10.8	**0.167** (8.0% ↓)	1.0 (90.3% ↓)	0.232	17.9	0.215 (7.2% ↓)	1.8 (90.0% ↓)	0.266	10.6	0.258 (2.9% ↓)	1.1 (90.0% ↓)	0.375	11.4	0.349 (7.0% ↓)	1.1 (90.2% ↓)	0.380	10.6	0.198 (47.8% ↓)	1.0 (90.1% ↓)
	192	0.247	10.6	**0.220** (11.0% ↓)	1.0 (90.5% ↓)	0.281	17.9	0.280 (0.4% ↓)	1.8 (90.0% ↓)	0.304	10.6	0.306 (0.7% ↑)	1.0 (90.4% ↓)	0.512	11.4	0.469 (8.4% ↓)	2.6 (76.8% ↓)	0.584	10.6	0.282 (51.6% ↓)	1.0 (90.1% ↓)
	336	0.329	10.6	**0.293** (10.9% ↓)	1.0 (90.1% ↓)	0.355	17.9	0.337 (5.1% ↓)	1.8 (90.0% ↓)	0.360	10.6	0.375 (4.1% ↑)	1.0 (90.1% ↓)	0.604	11.4	0.560 (7.3% ↓)	4.2 (63.0% ↓)	0.684	10.6	0.343 (49.9% ↓)	1.0 (90.1% ↓)
	720	0.410	10.6	**0.368** (10.1% ↓)	1.0 (90.2% ↓)	0.410	17.9	0.410 (0.0% ↑)	1.8 (90.1% ↓)	0.423	10.6	0.416 (1.7% ↓)	2.1 (80.3% ↓)	0.987	11.4	1.400 (41.8% ↑)	9.4 (17.6% ↓)	0.930	10.6	0.476 (48.9% ↓)	1.0 (90.4% ↓)

Table B.2: Univariate prediction comparison on the ETTm2 and Exchange datasets for prediction length $H \in \{96, 192, 336, 720\}$. For each model, the performance of the original (Dense) and the pruned model using PALS is presented in terms of MSE and parameter count. The difference between the results compared to the dense counterpart is shown in parenthesis as % (blue indicates improvement in the results compared to the dense model). **Bold** entries are the best performer in each experiment among various models.

Datasets	H	NSTransformer				FEDformer				Autoformer				Informer				Transformer			
		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS	
		MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param
ETTm2	96	0.074	10.6	0.068 (7.6% ↓)	1.1 (90.0% ↓)	0.069	17.8	**0.065** (5.9% ↓)	1.7 (90.2% ↓)	0.125	10.5	0.127 (1.7% ↑)	2.5 (76.6% ↓)	0.092	11.3	0.092 (0.2% ↑)	2.7 (76.5% ↓)	0.079	10.5	0.070 (12.0% ↓)	1.1 (89.3% ↓)
	192	0.128	10.7	0.107 (16.3% ↓)	1.1 (90.2% ↓)	**0.100**	17.8	0.101 (0.6% ↑)	1.7 (90.2% ↓)	0.141	10.5	0.144 (2.2% ↑)	1.0 (90.4% ↓)	0.137	11.3	0.129 (6.0% ↓)	1.1 (90.1% ↓)	0.119	10.5	0.117 (1.3% ↓)	7.0 (33.8% ↓)
	336	0.146	10.5	0.153 (5.4% ↑)	1.0 (90.1% ↓)	0.133	17.8	**0.131** (1.6% ↓)	2.6 (85.7% ↓)	0.146	10.5	0.143 (2.4% ↓)	1.0 (90.4% ↓)	0.174	11.3	0.170 (2.5% ↓)	5.2 (54.3% ↓)	0.171	10.5	0.137 (19.5% ↓)	1.0 (90.1% ↓)
	720	0.225	10.5	0.230 (2.0% ↑)	3.7 (65.3% ↓)	0.185	17.8	0.186 (0.2% ↑)	2.6 (85.4% ↓)	0.195	10.5	**0.181** (7.3% ↓)	1.0 (90.3% ↓)	0.211	11.3	0.213 (1.0% ↑)	6.3 (44.6% ↓)	0.192	10.5	0.189 (1.7% ↓)	6.0 (42.6% ↓)
Exchange	96	0.161	10.5	0.150 (6.8% ↓)	8.4 (20.0% ↓)	**0.122**	17.8	0.131 (7.4% ↑)	12.2 (31.4% ↓)	0.161	10.5	0.159 (1.2% ↓)	8.2 (21.8% ↓)	0.374	11.3	0.316 (15.6% ↓)	6.9 (39.2% ↓)	0.329	10.5	0.268 (18.4% ↓)	8.6 (18.5% ↓)
	192	**0.222**	10.5	0.252 (13.7% ↑)	8.5 (19.1% ↓)	0.257	17.8	0.276 (7.3% ↑)	9.2 (48.4% ↓)	0.304	10.5	0.345 (13.5% ↑)	8.1 (23.2% ↓)	1.180	11.3	1.064 (9.8% ↓)	8.4 (25.3% ↓)	1.549	10.5	1.339 (13.5% ↓)	8.0 (23.8% ↓)
	336	0.395	10.5	**0.327** (17.3% ↓)	8.6 (18.3% ↓)	0.499	17.8	0.525 (5.2% ↑)	13.8 (22.9% ↓)	0.669	10.5	0.616 (8.0% ↓)	7.4 (29.8% ↓)	1.771	11.3	1.747 (1.4% ↓)	9.2 (18.3% ↓)	2.822	10.5	2.115 (25.1% ↓)	8.2 (21.8% ↓)
	720	0.981	10.5	**0.973** (0.8% ↓)	8.2 (22.1% ↓)	1.258	17.8	1.295 (2.9% ↑)	13.7 (23.0% ↓)	1.284	10.5	1.311 (2.1% ↑)	7.0 (33.7% ↓)	1.497	11.3	1.764 (17.9% ↑)	9.1 (19.9% ↓)	2.091	10.5	2.226 (6.4% ↑)	8.4 (20.5% ↓)

Table B.3: Summary of the results on the ETTm2 and Exchange datasets in Table B.2. For each experiment on a transformer model and dataset, the average MSE, MAE, and number of parameters ($\times 10^6$) for various prediction lengths are reported before and after applying PALS. The difference between these results is shown in % where the blue color means improvement of PALS compared to the corresponding dense model.

Model	ETTm2-uni			Exchange-uni		
	MSE	MAE	#Params	MSE	MAE	#Params
NSTransformer	0.143	0.285	10.6	0.440	0.485	10.5
+PALS	0.140	0.277	1.7	**0.425**	**0.470**	8.4
Difference	2.4% ↓	2.6% ↓	83.9% ↓	3.3% ↓	3.1% ↓	19.8% ↓
FEDformer	0.122	0.265	17.8	0.534	0.520	17.8
+PALS	**0.121**	**0.262**	2.2	0.557	0.531	12.2
Difference	1.1% ↓	1.0% ↓	87.9% ↓	4.2% ↑	2.2% ↑	31.5% ↓
Autoformer	0.152	0.300	10.5	0.605	0.561	10.5
+PALS	0.149	0.297	**1.4**	0.608	0.561	**7.7**
Difference	2.1% ↓	0.9% ↓	86.9% ↓	0.5% ↑	0.1% ↓	27.1% ↓
Informer	0.153	0.304	11.3	1.206	0.852	11.3
+PALS	0.151	0.303	3.8	1.223	0.866	8.4
Difference	1.7% ↓	0.4% ↓	66.4% ↓	1.4% ↑	1.6% ↑	25.7% ↓
Transformer	0.140	0.286	10.5	1.698	0.928	10.5
+PALS	0.128	0.274	3.8	1.487	0.887	8.3
Difference	8.5% ↓	4.2% ↓	63.9% ↓	12.4% ↓	4.4% ↓	21.2% ↓

B.5 Hyperparameter Sensitivity Analysis

In this Appendix, we present the results for the hyperparameter sensitivity of PALS. We vary the values of γ and λ in $\{1.05, 1.1, 1.2\}$. The results are presented in Table B.4 and discussed in Section 3.6.2.

B.6 Pruning DLinear with PALS

In this Appendix, we train PALS with the DLinear [ZCZX22] model which is an MLP-based model for time series forecasting and has proven to be effective across various datasets. While our focus in this work is to reduce the complexity of transformers for time series forecasting, we want to show the generality of our proposed approach to other models. We demonstrate that PALS can be also applied to these models (which are computationally cheaper than transformers) to decrease the model size even more. As an example, we apply PALS to DLinear.

In Table B.5, the results of applying PALS to DLinear in terms of MSE and the sparsity level are presented. In most cases considered, PALS can prune DLinear without compromising loss. On the Electricity, ETTm2, Traffic, and Weather

Table B.4: Hyperparameter Sensitivity of PALS. The prediction MSE and model's parameter count ($\times 10^6$) when applying PALS on NSTransformers is measured when changing the values of γ and λ in $\{1.05, 1.1, 1.2\}$. Blue indicates improvement in the results compared to the dense model. **Bold** entries are the best performer in each row.

			$\gamma = 1.05$			$\gamma = 1.1$			$\gamma = 1.2$		
	H	Dense	$\lambda = 1.05$	$\lambda = 1.1$	$\lambda = 1.2$	$\lambda = 1.05$	$\lambda = 1.1$	$\lambda = 1.2$	$\lambda = 1.05$	$\lambda = 1.1$	$\lambda = 1.2$
Electricity	96	**0.170**	0.195 (78.2%)	0.193 (74.8%)	0.193 (74.8%)	0.200 (90.3%)	0.198 (90.1%)	0.199 (90.1%)	0.206 (90.6%)	0.206 (90.5%)	0.206 (90.2%)
	192	**0.186**	0.204 (84.3%)	0.211 (78.0%)	0.209 (78.6%)	0.215 (90.1%)	0.214 (90.1%)	0.214 (90.1%)	0.258 (90.3%)	0.260 (90.6%)	0.260 (90.3%)
	336	**0.198**	0.215 (90.0%)	0.218 (90.0%)	0.219 (90.0%)	0.243 (90.1%)	0.246 (90.1%)	0.244 (90.1%)	0.271 (90.6%)	0.271 (90.4%)	0.270 (90.2%)
	720	**0.222**	0.240 (78.1%)	0.242 (73.5%)	0.242 (73.5%)	0.250 (90.1%)	0.251 (90.1%)	0.252 (90.0%)	0.297 (90.4%)	0.295 (90.3%)	0.300 (90.1%)
ETTm2	96	0.241	0.272 (25.7%)	0.263 (30.3%)	0.268 (64.6%)	0.244 (40.0%)	0.248 (61.0%)	0.202 (90.4%)	**0.193** (90.3%)	0.193 (90.3%)	0.195 (90.1%)
	192	**0.387**	0.644 (21.4%)	0.626 (21.5%)	0.650 (25.9%)	0.574 (27.4%)	0.547 (41.1%)	0.498 (54.5%)	0.541 (40.4%)	0.393 (65.4%)	0.484 (65.3%)
	336	0.615	0.529 (20.5%)	0.529 (21.2%)	0.480 (27.9%)	0.753 (21.9%)	0.725 (35.7%)	0.563 (67.5%)	0.473 (73.7%)	**0.340** (90.2%)	0.340 (90.1%)
	720	0.734	0.751 (19.7%)	0.780 (27.7%)	0.793 (42.1%)	0.777 (30.9%)	0.804 (34.9%)	0.641 (78.5%)	0.678 (45.5%)	0.600 (79.2%)	**0.441** (90.2%)
Exchange	96	0.124	0.139 (21.1%)	0.138 (22.3%)	0.135 (32.5%)	0.139 (22.3%)	0.143 (22.5%)	0.139 (64.2%)	0.137 (28.9%)	**0.113** (79.5%)	0.125 (87.0%)
	192	0.243	0.269 (16.3%)	0.269 (16.3%)	0.269 (16.3%)	0.266 (22.9%)	0.264 (27.7%)	0.263 (30.1%)	0.249 (34.6%)	0.241 (38.1%)	**0.239** (60.9%)
	336	0.462	0.486 (17.7%)	0.486 (17.7%)	0.486 (17.7%)	0.459 (22.0%)	0.459 (22.0%)	0.459 (22.0%)	**0.429** (13.1%)	0.478 (18.3%)	0.460 (28.7%)
	720	1.336	1.263 (17.0%)	1.263 (17.0%)	1.263 (17.0%)	**1.134** (22.9%)	**1.134** (22.9%)	**1.134** (22.9%)	1.236 (15.6%)	1.236 (15.6%)	1.236 (15.6%)
Illness	24	**2.555**	2.593 (27.5%)	2.638 (34.3%)	2.603 (31.6%)	2.668 (44.5%)	2.620 (47.9%)	2.613 (53.0%)	2.849 (65.6%)	2.784 (70.9%)	2.884 (75.7%)
	36	**1.913**	2.112 (24.8%)	2.118 (27.5%)	2.125 (30.8%)	2.193 (34.3%)	2.220 (41.3%)	2.246 (51.6%)	2.433 (53.0%)	2.496 (59.7%)	2.644 (70.9%)
	48	**1.873**	2.186 (25.6%)	2.207 (31.3%)	2.214 (34.0%)	2.246 (39.6%)	2.277 (48.8%)	2.332 (53.1%)	2.492 (60.4%)	2.650 (65.4%)	2.632 (63.5%)
	60	**2.209**	2.413 (30.7%)	2.425 (34.5%)	2.425 (34.5%)	2.523 (50.9%)	2.549 (56.4%)	2.577 (48.0%)	2.772 (57.8%)	2.895 (64.7%)	3.019 (58.6%)
Traffic	96	**0.608**	0.641 (69.6%)	0.647 (68.9%)	0.647 (68.9%)	0.676 (70.6%)	0.666 (90.1%)	0.681 (80.3%)	0.671 (90.4%)	0.669 (90.2%)	0.674 (90.6%)
	192	**0.623**	0.658 (75.7%)	0.661 (69.6%)	0.661 (59.9%)	0.688 (76.4%)	0.694 (80.2%)	0.720 (60.2%)	0.711 (90.1%)	0.697 (90.1%)	0.693 (90.4%)
	336	**0.640**	0.672 (78.7%)	0.671 (76.7%)	0.675 (72.9%)	0.712 (76.9%)	0.688 (90.3%)	0.690 (90.3%)	0.715 (90.4%)	0.720 (90.1%)	0.718 (90.6%)
	720	**0.658**	0.687 (62.4%)	0.687 (62.4%)	0.687 (62.4%)	0.730 (76.4%)	0.735 (68.8%)	0.735 (68.8%)	0.728 (90.3%)	0.721 (90.3%)	0.724 (90.9%)
Weather	96	0.182	**0.166** (78.2%)	0.167 (80.8%)	0.167 (80.8%)	0.167 (90.3%)	0.167 (90.5%)	0.167 (90.5%)	0.172 (90.2%)	0.172 (90.1%)	0.172 (90.1%)
	192	0.247	0.225 (83.7%)	0.224 (85.4%)	0.224 (85.4%)	0.222 (90.5%)	0.222 (90.5%)	0.222 (90.5%)	**0.220** (90.1%)	**0.220** (90.1%)	**0.220** (90.1%)
	336	0.329	0.303 (80.0%)	0.301 (80.4%)	0.301 (80.4%)	**0.294** (90.3%)	0.295 (90.5%)	0.295 (90.5%)	0.297 (90.2%)	0.298 (90.1%)	0.298 (90.1%)
	720	0.410	0.413 (68.9%)	0.392 (85.2%)	0.392 (85.2%)	0.372 (90.3%)	0.382 (90.5%)	0.382 (90.5%)	**0.364** (90.3%)	0.375 (90.1%)	0.375 (90.1%)

datasets, PALS can prune ∼ 90% of the connections while achieving comparable loss. This shows the effectiveness of PALS when applied to an MLP-based model.

Finally, we want to highlight that our goal in this work is not to propose a new forecasting model and beat the state-of-the-art for time series forecasting. Instead, we aim to decrease the high computational costs of models for time series forecasting while finding automatically a decent sparsity level and potentially improving their generalization. The reason that we focus on transformers is that they are considered to be computationally expensive while performing well in time series forecasting, and as shown in [WHL+22], transformer-based models perform generally well in other time series analysis tasks, including, classification, anomaly detection, and imputation compared to the MLP-based models [ZCZX22]. Therefore, they can be a promising direction for future time series analysis research. However, PALS is orthogonal to forecasting models and can be applied to any deep learning-based model to reduce its computational costs.

Table B.5: Effectiveness of PALS for pruning DLinear model [ZCZX22]. Each row presents the results of DLinear before and after applying PALS in terms of MSE, for each prediction length. The achieved sparsity level is shown in parenthesis as %.

	Model	Electricity	ETTm2	Exchange	Illness	Traffic	Weather
96/24	**DLinear**	0.140	0.172	0.094	1.997	0.413	0.175
	+PALS	0.141 (90.2%)	0.181 (90.0%)	0.086 (25.5%)	1.984 (34.1%)	0.412 (90.3%)	0.177 (90.2%)
192/36	**DLinear**	0.153	0.235	0.168	2.090	0.424	0.217
	+PALS	0.154 (90.2%)	0.249 (90.1%)	0.173 (20.8%)	2.130 (29.8%)	0.424 (90.2%)	0.218 (90.2%)
336/48	**DLinear**	0.169	0.307	0.322	2.058	0.437	0.263
	+PALS	0.170 (90.2%)	0.301 (82.7%)	0.324 (24.7%)	2.124 (32.6%)	0.436 (90.1%)	0.262 (90.2%)
720/60	**DLinear**	0.204	0.390	0.959	2.375	0.467	0.328
	+PALS	0.204 (90.1%)	0.433 (90.3%)	0.963 (38.0%)	2.358 (44.3%)	0.467 (90.2%)	0.324 (90.2%)

B.7 Efficiency of PALS

In this appendix, we discuss the efficiency of PALS in terms of computational costs from various perspectives.

B.7.1 Training FLOPs

In this section, we present a comprehensive analysis of the training computational efficiency, as outlined in Table B.6. By looking into the inference FLOPs in Table 3.3 in Chapter 3, we can observe that the efficiency gain during training is

even higher than inference. This observation underscores the significance of optimizing computational resources for training, a critical aspect in the deployment of machine learning algorithms.

B.7.2 Pruning Capabilities

Based on the observations in Section 3.6.1, PALS achieves the highest average sparsity level among the considered pruning and sparse training methods. More importantly, it finds the optimal sparsity level automatically without requiring any prior information, while most pruning and sparse training algorithms need to receive the sparsity level as an input of the algorithm. In short, PALS can find a network with higher sparsity than the competitors (GMP, GraNet, and RigL), where the sparsity level is found automatically.

B.7.3 Convergence Speed

Another factor that we consider regarding the efficiency of the methods is the convergence speed. If a method converges faster than the others, it can be considered to be more efficient in terms of resource usage.

To compare the convergence speed of each model, we compare the training epochs. As we use early stopping during training, each training round might not need the full training epochs (which is set to 10). In Table B.7, we report the average number of training epochs for various datasets. The results are an average for different prediction lengths and three random seeds. While GraNet needs almost full training time due to using a fixed pruning schedule which is determined based on the number of epochs, PALS can automate the pruning speed based on the loss. On four out of six datasets, PALS converges faster than GraNet and the dense model. On the Weather dataset, PALS requires longer training time than the dense model. As will be explained in Section B.7.4, PALS performs most of this training at a very high sparsity level ($\sim 90\%$), thus being resource-efficient. Overall, PALS can achieve a higher or comparable speed to the dense model in most cases considered.

B.7.4 Sparsity During Training

Another factor affecting the efficiency of PALS is the sparsity during training. As in our experiments, PALS starts from a dense network, we analyze when it reaches the final sparsity and how the sparsity changes during training. Then, we

Table B.6: Summary of the results on the benchmark Datasets in Table B.1. For each experiment on a transformer model and dataset, the average MSE, MAE, number of parameters ($\times 10^6$), and the training FLOPs count ($\times 10^{12}$) for various prediction lengths are reported before and after applying PALS. The difference between these results is shown in % where the blue color means improvement of PALS compared to the corresponding dense model.

Model	Electricity				ETTm2				Exchange				Illness				Traffic				Weather			
	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs
NSTransformer	**0.19**	**0.30**	12.0	898.95	0.49	0.43	10.6	854.73	0.54	0.49	10.6	113.93	**2.14**	**0.92**	10.5	6.21	0.63	**0.34**	14.2	721.70	0.29	0.31	10.7	810.94
+PALS	0.21	0.32	2.2	27.11	0.38	0.39	2.5	41.99	**0.49**	**0.47**	**5.4**	51.50	2.33	0.97	7.4	1.44	0.67	0.37	4.3	35.29	**0.26**	**0.29**	1.0	**21.24**
Difference	10.8% ↑	7.3% ↑	81.5% ↓	97.0% ↓	24.0% ↓	11.2% ↓	76.7% ↓	95.1% ↓	9.3% ↓	3.6% ↓	48.5% ↓	54.8% ↓	9.1% ↑	5.1% ↑	30.0% ↓	76.8% ↓	5.2% ↑	9.1% ↑	70.1% ↓	95.1% ↓	10.2% ↓	6.9% ↓	90.3% ↓	97.4% ↓
FEDformer	0.21	0.32	19.5	965.68	**0.30**	0.35	17.9	1094.02	0.50	0.49	17.9	151.60	2.84	1.14	13.7	6.64	**0.61**	0.38	22.3	618.66	0.32	0.37	17.9	1027.26
+PALS	0.23	0.34	3.0	21.09	0.30	**0.35**	1.8	20.92	0.51	0.50	10.5	**32.44**	3.05	1.19	8.3	**0.99**	0.62	0.38	5.6	30.39	0.31	0.36	1.8	22.73
Difference	9.0% ↑	4.9% ↑	84.7% ↓	97.8% ↓	1.5% ↑	0.5% ↓	90.2% ↓	98.1% ↓	2.1% ↑	1.1% ↑	41.2% ↓	78.6% ↓	7.2% ↑	5.0% ↑	39.5% ↓	85.1% ↓	1.0% ↑	1.1% ↑	74.6% ↓	95.1% ↓	2.8% ↓	3.2% ↓	90.0% ↓	97.8% ↓
Autoformer	0.24	0.34	12.1	751.01	0.33	0.37	10.5	867.75	0.58	0.53	10.5	111.87	3.08	1.18	10.5	5.89	0.64	0.40	14.9	669.03	0.34	0.38	10.6	1175.29
+PALS	0.26	0.36	2.7	27.92	0.31	0.35	**1.0**	**20.23**	0.62	0.55	7.1	40.54	3.19	1.22	**6.7**	1.17	0.65	0.41	4.5	31.83	0.34	0.38	1.3	32.32
Difference	9.3% ↑	4.6% ↑	77.7% ↓	96.3% ↓	8.1% ↓	5.6% ↓	90.3% ↓	97.7% ↓	5.5% ↑	4.4% ↑	32.7% ↓	63.8% ↓	3.5% ↑	3.2% ↑	36.6% ↓	80.2% ↓	1.9% ↑	2.1% ↑	69.5% ↓	95.2% ↓	0.1% ↑	1.3% ↓	87.7% ↓	97.3% ↓
Informer	0.36	0.43	12.5	917.53	1.53	0.88	11.3	694.06	1.59	1.00	11.3	135.66	5.27	1.58	11.3	5.23	0.81	0.46	14.4	683.57	0.62	0.55	11.4	1135.86
+PALS	0.42	0.48	**1.4**	**14.93**	1.39	0.83	5.3	187.17	1.53	0.98	8.6	59.27	5.23	1.57	7.4	1.15	0.94	0.53	**2.3**	**19.08**	0.69	0.56	4.3	187.78
Difference	18.9% ↑	11.3% ↑	88.6% ↓	98.4% ↓	9.0% ↓	5.8% ↓	53.4% ↓	73.0% ↓	3.9% ↓	1.6% ↓	24.2% ↓	56.3% ↓	0.8% ↓	1.0% ↓	34.9% ↓	78.0% ↓	15.5% ↑	15.3% ↑	83.9% ↓	97.2% ↓	12.1% ↑	2.0% ↑	61.9% ↓	83.5% ↓
Transformer	0.28	0.38	11.7	868.54	1.48	0.86	10.5	922.01	1.61	0.97	10.5	134.00	4.94	1.49	10.5	6.43	0.67	0.36	13.6	630.91	0.64	0.56	10.6	874.91
+PALS	0.31	0.40	2.5	31.28	1.08	0.75	3.2	205.35	1.41	0.91	6.6	34.74	4.91	1.48	7.7	1.49	0.69	0.38	3.8	29.67	0.32	0.38	**1.0**	21.53
Difference	10.5% ↑	5.7% ↑	78.3% ↓	96.4% ↓	26.7% ↓	13.3% ↓	69.9% ↓	77.7% ↓	12.5% ↓	6.1% ↓	37.5% ↓	74.1% ↓	0.7% ↓	0.9% ↓	27.2% ↓	76.8% ↓	3.3% ↑	5.4% ↑	71.7% ↓	95.3% ↓	49.6% ↓	32.5% ↓	90.2% ↓	97.5% ↓

Table B.7: Comparison of convergence speed in terms of the number of training epochs on the NSTransformer model. The results are average over various prediction lengths of $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$).

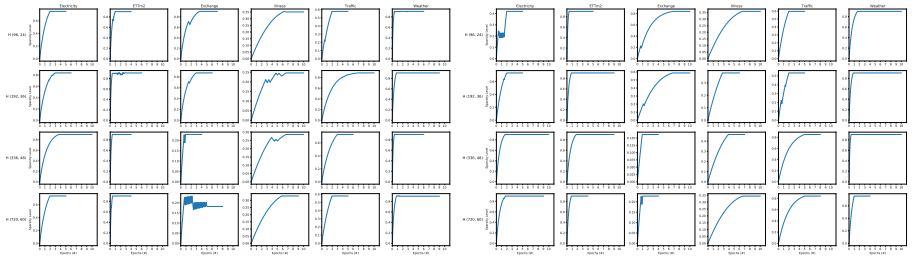
Dataset	PALS	GraNet	Dense
Electricity	**8.83**	9.75	8.92
ETTm2	**4.58**	9.00	4.92
Exchange	5.83	9.42	**4.33**
Illness	**7.97**	9.58	8.92
Traffic	**8.83**	9.50	9.5
Weather	7.00	9.08	**4.08**

would be able to analyze whether the forward pass is mostly performed sparsely or not.

To achieve this, we plot the sparsity level during the training of PALS for each transformer model and dataset for various prediction lengths. The sparsity levels are measured after each pruning iteration which is repeated every 5 batches on the Illness dataset and every 20 batches on the other datasets. These values are measured till the last pruning iteration before saving the model. The results for all models are presented in Figure B.2.

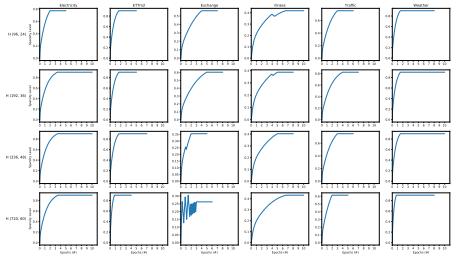
In Figure B.2, due to different convergence speeds on each dataset, each model requires a different number of training epochs. As the pruning is tuned based on the loss value, the pruning speed varies for each dataset and model. For example, on the Weather, Electricity, and Ettm2 datasets, for most of the considered transformer models and prediction lengths, the models reach the final sparsity level within a few training epochs. However, for the Exchange, Illness, and Traffic datasets, the convergence speed can be different among different models and prediction lengths. Also, in these datasets, the final sparsity is reached slower than the earlier category (Weather, Electricity, and Ettm2 datasets). However, in most cases considered, the final sparsity is reached only within half of the training period.

Overall, it can be observed that in most cases PALS reaches the final sparsity level in a few epochs. Therefore, the forward pass during training is performed sparsely for a large fraction of the training process.

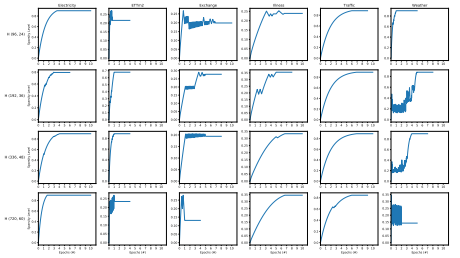


(a) NSTransformer.

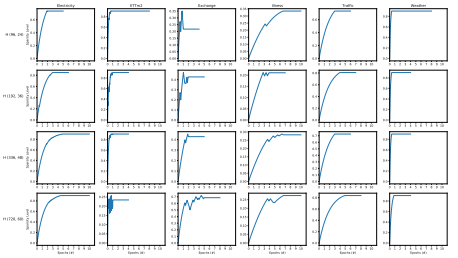
(b) Autoformer.



(c) FEDformer.



(d) Informer.



(e) Transformer.

Figure B.2: Sparsity level of each network during training of PALS. In most cases, the final sparsity is achieved within a few epochs after the training starts. Therefore, the forward pass during training is performed sparsely for a large fraction of the training process.

B.7.5 Sparse Implementation.

Another aspect regarding efficiency is concerned with the true sparse implementation. Further steps are required to achieve the full advantage of GPU speedup. However, this is a challenging research question that is out of the scope of the

current work. Unstructured sparsity (e.g., pruning connections of a network as we use in this work) still suffers from hardware support [LW23], while proven to be more effective than structured sparsity (e.g., pruning neurons or other units) [HABN+21]. Although there have been some works that have implemented truly sparse neural networks efficiently on CPUs [LCC+21, CMP21, ASvdL+22] and just for MLPs and Autoencoders, proper and efficient GPU support for sparse matrix computation is yet to be developed. In this work, we focus on the algorithmic aspect of developing resource-efficient algorithms, while getting full advantage of sparsity requires a large amount of research that we cannot address with our current human resources. We hope that works such as ours can light an awareness signal, bringing more researchers from the community together to start addressing seriously the "elephant in the room".

B.8 Model Size Effect

In this Appendix, we study the effect of model size on the performance of PALS by changing the hidden dimension (d_{model}) in {256, 512 (default), 768} to analyze the trade-off between loss and parameters count. We have also considered the results of the original dense model ($d_{model} = 512$). The results for all models in terms of MSE and parameters count (which are the average results over different prediction lengths) are presented in Figure B.3.

In Figure B.3, we can observe that in most cases considered, the model can be pruned significantly with little or no increase in loss. Similar to our discussion in Section 3.3, we see consistent behavior among the Electricity, Illness, and traffic datasets, where a higher number of parameters is mostly in favor of loss reduction. This might come from the inherent complexity of predicting these datasets due to either a high number of variables (such as Electricity and Traffic datasets) or the relatively non-stationary nature of data (such as the illness datasets as shown in [LWWL22]). While being a non-stationary dataset, on the Exchange dataset, a small ($d_{model} = 256$) sparse model can perform very closely or better than the original ($d_{model} = 512$) dense model on average among various transformer architectures. This might be caused by the abilities of the transformer variants to learn this behavior such that even a pruned small model can substitute this model. On the other datasets (ETTm2 and Weather which are relatively stationary with a low number of variables), sparsity results mostly in a better performance than the dense model.

In short, it can be concluded that various time-series datasets do not demonstrate homogenous behavior due to their intrinsic differences. Therefore, we

need to consider these differences when choosing the model size or the right sparsity level (Section 3.3).

B.9 Prediction Quality

In this Appendix, we evaluate the prediction of different models and discuss how PALS affects this prediction when pruning them.

The predictions for the transformer on the Weather, ETTm2, and Exchange datasets, where the most significant changes in loss and parameters count were seen, are visualized in Figure B.4. It can be observed that PALS significantly improves the prediction of this model on these three datasets. As summarized in

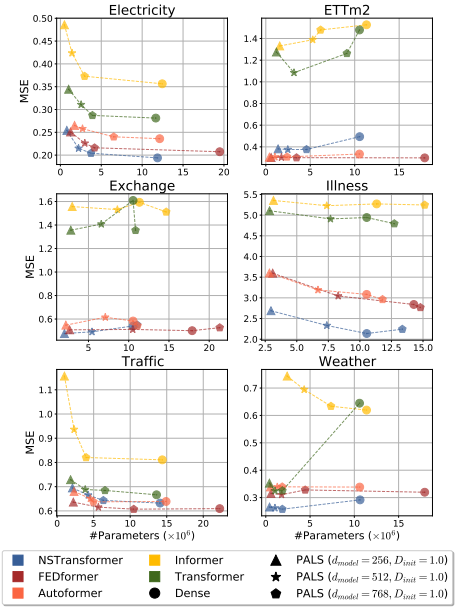
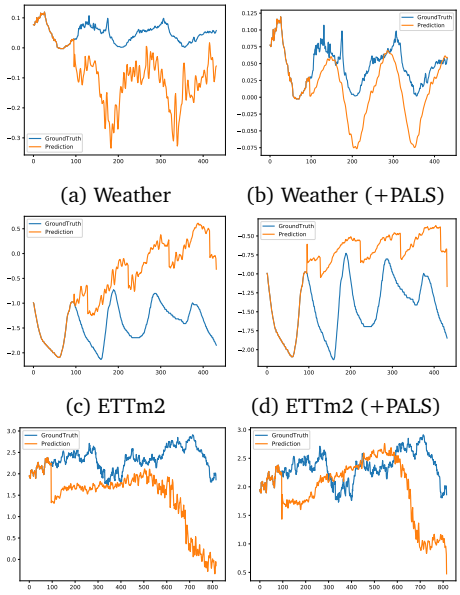


Figure B.3: Model size effect by varying $d_{model} \in \{256, 512(default), 768\}$) on the prediction performance of PALS compared to the original dense model.



(a) Weather (b) Weather (+PALS)

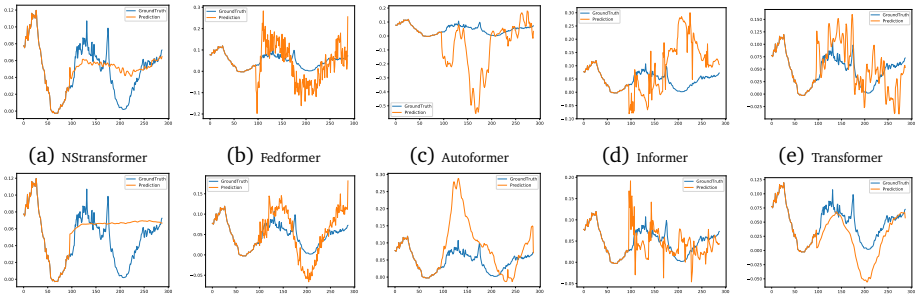
(c) ETTm2 (d) ETTm2 (+PALS)

(e) Exchange (f) Exchange (+PALS)

Figure B.4: Forecasting Visualization on Transformer with and without PALS on three datasets.

Table B.1 (Appendix B.3), PALS is able to gain very close prediction performance on the transformer to the best performer on the Weather dataset (NStransformer) while pruning 90% of unimportant connections.

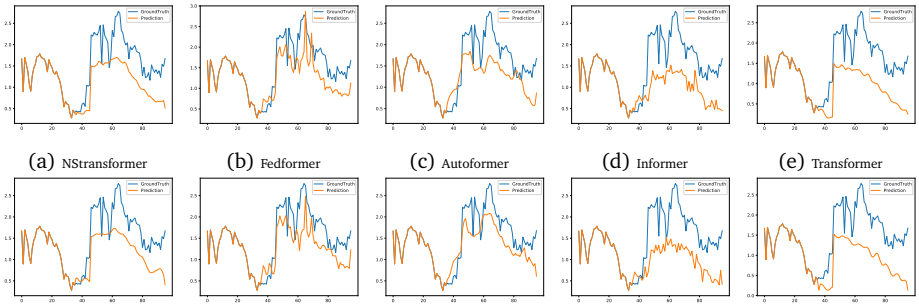
Next, we visualize the predictions for each model with and without PALS on the Weather, Illness, ETTm2 (periodic dataset), and Exchange (without obvious periodicity) datasets for all transformer variants in Figures B.5, B.6, B.7, and B.8, respectively. In most cases considered in these figures, PALS is able to gain very similar or better performance than the dense counterpart model. By removing unimportant connections, the prediction using PALS is mostly smoother than the prediction of the dense model. Therefore, it can be concluded that it is possible to significantly prune transformers for time series forecasting using PALS without compromising performance.



(a) NStransformer (b) Fedformer (c) Autoformer (d) Informer (e) Transformer

(f) NStransformer + PALS (g) Fedformer + PALS (h) Autoformer + PALS (i) Informer + PALS (j) Transformer + PALS

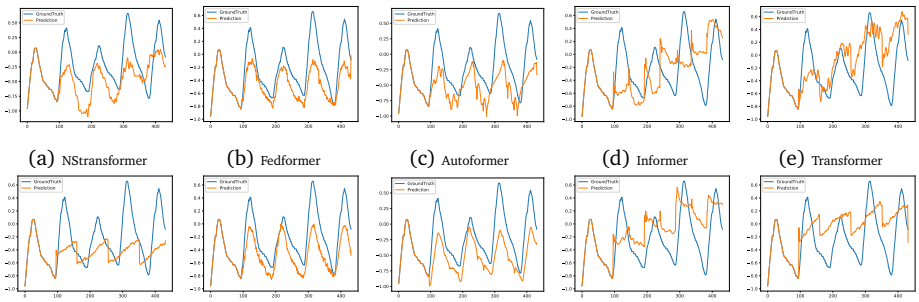
Figure B.5: Forecasting Visualization on transformers with and without PALS on the Weather dataset ($H = 192$).



(a) NStransformer (b) Fedformer (c) Autoformer (d) Informer (e) Transformer

(f) NStransformer + PALS (g) Fedformer + PALS (h) Autoformer + PALS (i) Informer + PALS (j) Transformer + PALS

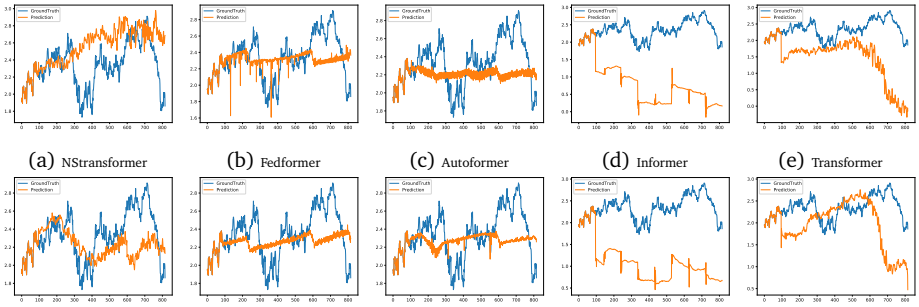
Figure B.6: Forecasting Visualization on transformers with and without PALS on the illness dataset ($H = 60$).



(a) NStransformer (b) Fedformer (c) Autoformer (d) Informer (e) Transformer

(f) NStransformer + PALS (g) Fedformer + PALS (h) Autoformer + PALS (i) Informer + PALS (j) Transformer + PALS

Figure B.7: Forecasting Visualization on transformers with and without PALS on the ETTm2 dataset ($H = 336$).



(a) NStransformer (b) Fedformer (c) Autoformer (d) Informer (e) Transformer

(f) NStransformer + PALS (g) Fedformer + PALS (h) Autoformer + PALS (i) Informer + PALS (j) Transformer + PALS

Figure B.8: Forecasting Visualization on transformers with and without PALS on the Exchange dataset ($H = 720$).

Appendix C

Additional Experiments and Analysis for Chapter 4

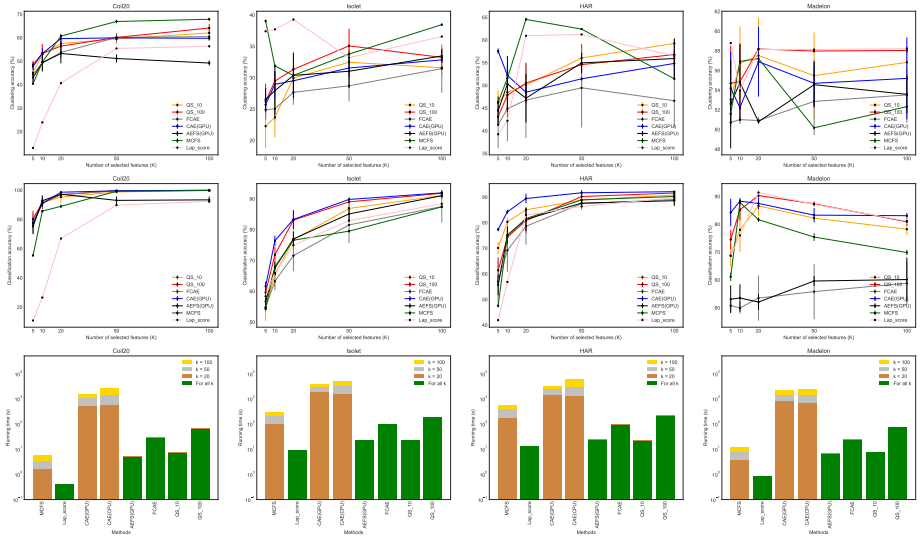
C.1 Performance Evaluation

In this appendix, we compare all methods from different aspects including accuracy, memory usage, running time, energy consumption, and the number of parameters. We perform different experiments to gain a deep insight into the performance of QuickSelection.

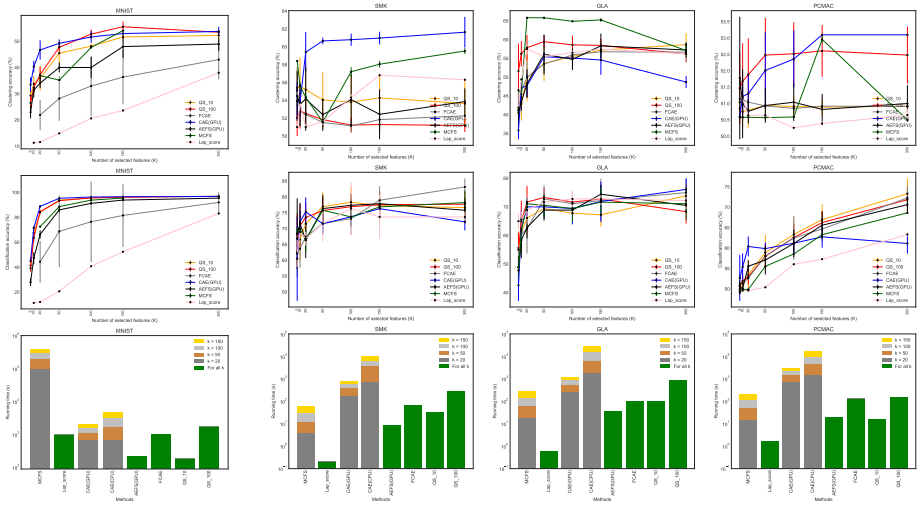
C.1.1 Discussion: Accuracy and Computational Efficiency Trade-off

In this section, we compare the performance of all methods in more detail. We run feature selection for different values of K on each dataset and then measure the performance.

As shown in Figure C.1, we compare clustering accuracy, classification accuracy, and running time among the methods for different values of K. The comparison of maximum memory (RAM) requirement is also depicted in Figure C.2. For all methods except CAE and AEFS, we run the experiments on a single CPU core. Since the implementations of CAE and AEFS are optimized for GPU, we measure the running time of these methods using a GPU. However, we also consider the running time of CAE using a single CPU core. It should be noticed



(a) Low-dimensional datasets



(b) MNIST (c) High-dimensional datasets

Figure C.1: Comparison of clustering accuracy, classification accuracy, and running time for various values of K among all the methods considered on eight datasets, including low-dimensional and high-dimensional datasets. The running time of CAE and AEFS is measured using a GPU, while all the other methods use only a single CPU core.

that since Laplacian score, AEFS, FCAE, and QuickSelection give the ranking of the features as the output of the feature selection process, we need to run them just once for all values of K. However, MCFS and CAE need the K value as an input of their algorithm. So, the running time depends on the value of K. In the implementation of AEFS, K is used to set the number of hidden values. However, it is not a requirement of the algorithm.

We summarize the results of the aforementioned plots in Figure C.3; we compare the methods using the *score 1*, which is introduced in Section 4.5.1. This score is computed based on the methods' ranking in clustering accuracy, classification accuracy, running time, and memory. As explained in Section 4.5.1, we give a score of one to each method that is the first or second-best performer in each of the considered metrics. Then, we compute a sum over all of these scores on all datasets and on all values of K; the final scores for each method can be seen in Figure C.3. The first column depicts the results on low-dimensional datasets with a low number of samples, including Coil20, Isolet, HAR, and Madelon. The second column shows the results corresponding to MNIST. Similarly, the third column corresponds to high-dimensional datasets, including SMK, GLA, and PCMAC. The total score over all of these datasets is shown in the 4^{th} column. In Figure C.3, there exist four rows; the first row corresponds to considering QuickSelection$_{10}$ and QuickSelection$_{100}$ simultaneously, and the sum of their scores are depicted in the second row. The last two rows correspond to considering each of these two methods separately.

However, since the performance of each method can be different in each of the three groups of datasets, we compute a normalized version of the score 1, based on the number of datasets in each group. For example, the Laplacian score has a poor performance on MNIST, and this pattern would be similar on other datasets with a large number of samples. However, there is just one dataset with a large number of samples in this experiment. On the other hand, on high-dimensional datasets with a low number of samples, this method has a good performance in terms of running time, and we have three datasets with such characteristics. So, we normalize the values of score 1, such that instead of giving a score of one to each method, we give a score of one divided by the number of datasets in the corresponding group. The results of the normalized score 1 are shown in the last column of Figure C.3.

C.1.2 Energy Consumption

We perform another experiment regarding the comparison of energy consumption among all methods. The results are presented in Figure C.4. More details

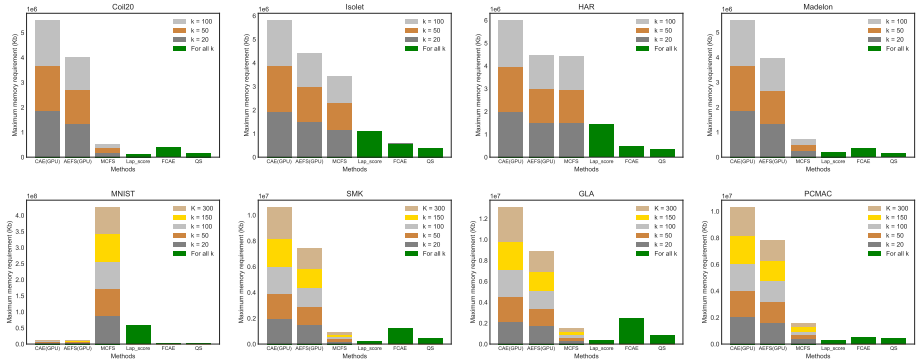


Figure C.2: Maximum memory usage during feature selection for different values of K.

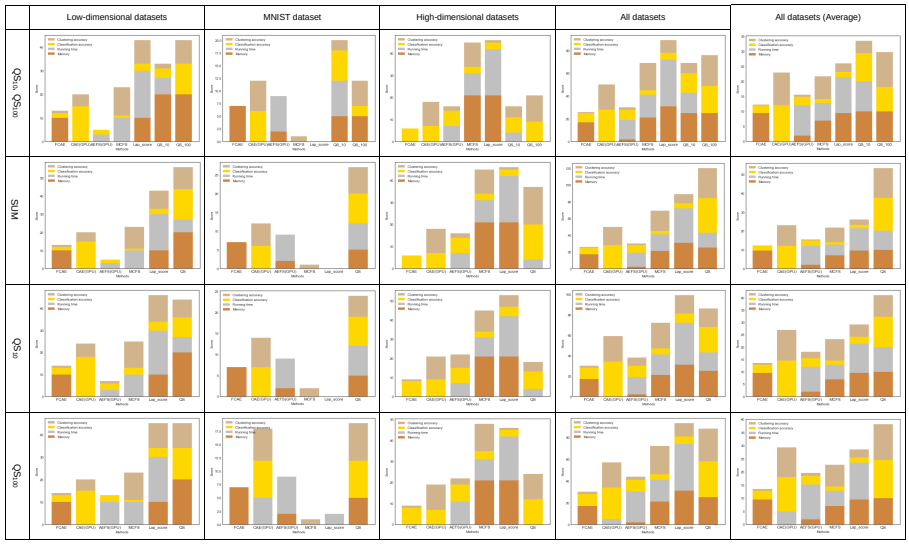


Figure C.3: Feature selection results comparison in terms of classification accuracy, clustering accuracy, speed, and memory. The Scores are based on the ranking of the methods on each dataset and for different values of K (Score 1).

regarding this plot are given in Chapter 4, Section 4.5.1.

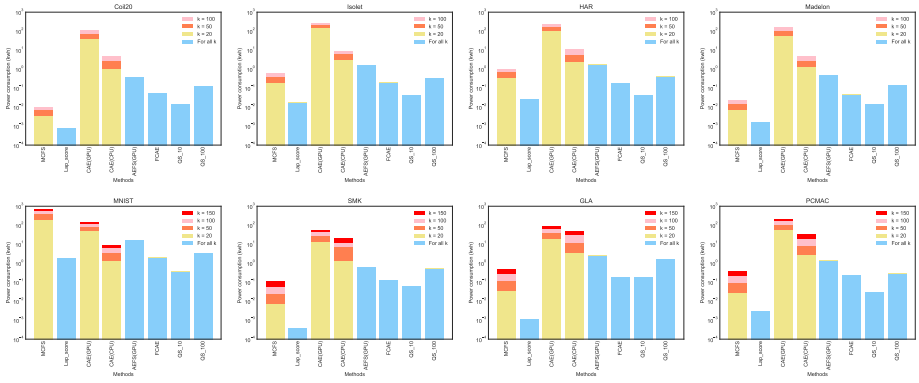


Figure C.4: Energy consumption of all methods for different values of K.

C.1.3 Number of Parameters

In Figure C.5, we compare the number of parameters of the autoencoder-based methods. FCAE, a fully connected-autoencoder with 1000 hidden neurons, has the highest number of parameters on all datasets. Our proposed network, sparse DAE, has the lowest number of parameters in most cases. It has 1000 hidden neurons that are sparsely connected to input and output neurons. The number of parameters of AEFS and CAE depends on the number of selected features. As also mentioned earlier, the structure of AEFS is similar to FCAE with a difference in the number of hidden neurons. The number of hidden neurons in the implementation of AEFS is set to K.

C.2 Parameter Selection

In this appendix, we discuss the effect of three hyperparameters of QuickSelection on feature selection performance.

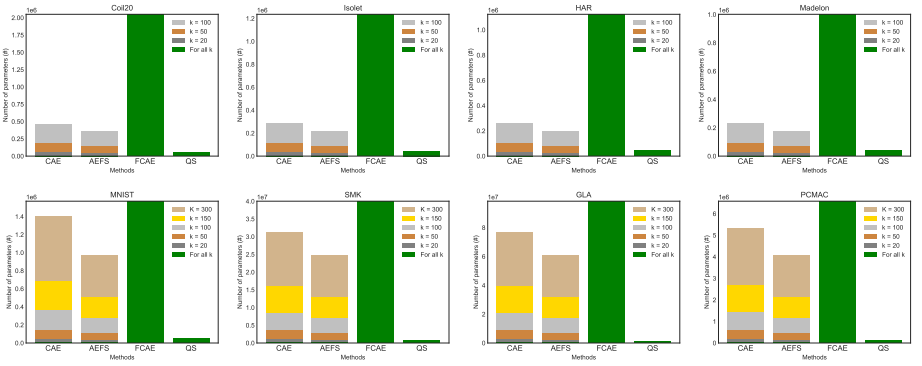


Figure C.5: Number of parameters of autoencoder-based models for different values of K.

C.2.1 Noise Factor

To analyze the effect of the noise level on QuickSelection behavior, we evaluate the sparse DAE model with different noise factors. To this end, we test different noise factors between 0 and 0.8. The results can be observed in Figure C.6. These results are an average of 5 runs for each case.

We can observe that adding 20% to 40% noise on the data seems to be optimal; it improves the performance on most of the datasets for QuickSelection$_{10}$ and QuickSelection$_{100}$ compared to the model without any noise. We choose the noise factor of 0.2 for all the experiments.

It is clear in Figure C.6, that setting the noise factor to a large value may corrupt the input data in such a way that the network would not be able to model the data distribution accurately. For example, on the Isolet dataset, the clustering accuracy degrades for 10% when we add 80% noise on the input data compared to the model with the noise factor of 0.2. Also, the result is less stable when we add a large amount of noise. In this example, we can observe that adding 20% noise to the original data improves both classification and clustering accuracy of QuickSelection$_{100}$ by approximately 3%.

From this figure, it can be observed that the improvement of adding noise, is more obvious in QuickSelection$_{100}$ than QuickSelection$_{10}$. When we add noise to the data, it needs more time to learn the original structure of the data. So, we need to run it for more epochs to get a proper result.

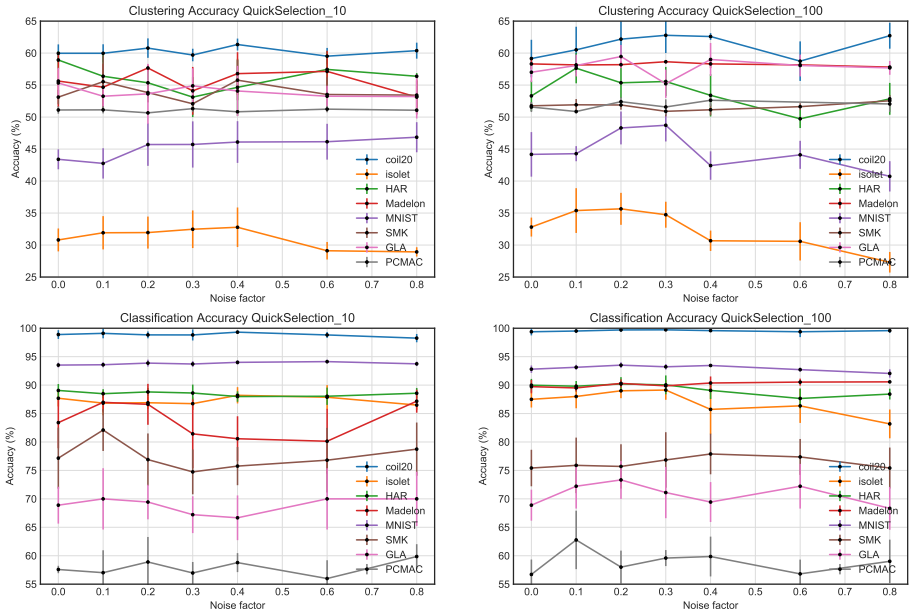


Figure C.6: Clustering and classification accuracy for feature selection using QuickSelection$_{10}$ and QuickSelection$_{100}$ with different values of noise factor. We select 50 features from all datasets except Madelon for which we select 20 features.

C.3 Visualization of Selected Features on MNIST

In Figure C.7, we visualize the 50 best features found by QuickSelection on the MNIST dataset at different epochs. These features are mostly at the center of the image, similar to the pattern of MNIST digits.

Then, we visualize the features selected for each class separately. In Figure C.8, each picture at different epochs is the average of the 50 selected features of all the samples of each class along with the average of the actual samples of the corresponding class. As we can see, during training, these features become more similar to the pattern of digits of each class. Thus, QuickSelection is able to find the most relevant features for all classes.

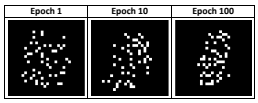


Figure C.7: 50 most informative features of MNIST dataset selected by QuickSelection after 1, 10, and 100 epochs of training.

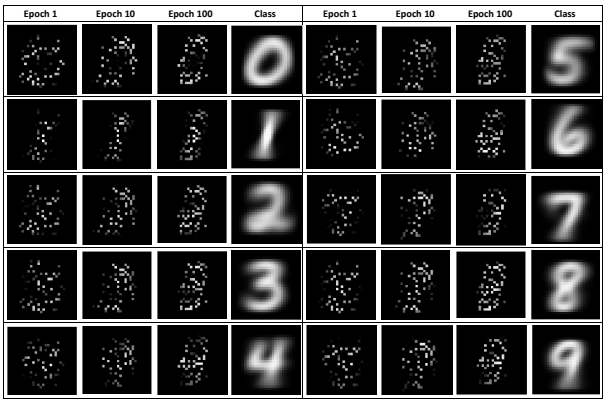


Figure C.8: Average of the data samples of each MNIST class corresponding to the 50 selected features after 1, 10, and 100 epochs of training along with the average of the actual samples of each class.

C.4 Feature Extraction

Although it is not the main focus of this work, we perform a small analysis on the MNIST dataset to study the performance of sparse DAE as a feature extractor. We train it to map the high-dimensional features into a lower-dimensional space.

The structure we consider for feature extraction has three hidden layers with 1000, 50, and 1000 neurons, respectively; the middle layer (50 neurons) is the extracted low-dimensional representation. We compare the results with fully-connected DAE (FC-DAE - implemented in Keras [C+15]). We also extract features using the Principal Component Analysis (PCA) [WEG87] technique as a baseline method. Then, we train an ExtraTrees classifier on these extracted features and compute the classification accuracy. The results are presented in Figure C.9.

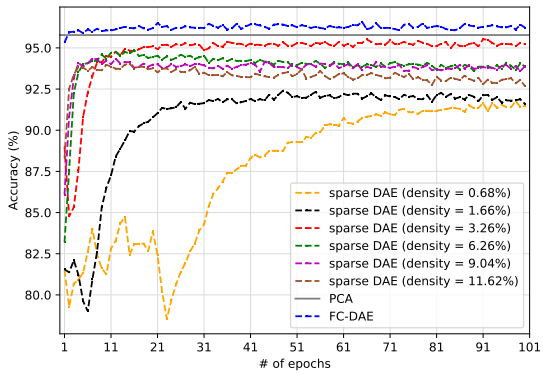


Figure C.9: Classification accuracy for feature extraction using sparse DAE with different density level on the MNIST dataset (number of extracted features = 50) compared with FC-DAE and PCA.

To achieve the best density level that suits our network, we test different ϵ values. As shown in Figure C.9, sparse DAE (density = 3.26%) has the best performance among them. Sparse DAE (density = 3.26%), FC-DAE, and PCA achieve 95.2%, 96.2%, and 95.6% accuracy, respectively. Although sparse DAE can not perform as well as the FC-DAE, it approximately has 54 k parameters compared to 1.67 m parameters of FC-DAE. Such a small number of parameters of this model results in a high rise in the running speed and a significant drop in the memory requirement. Furthermore, it is interesting to observe that a very sparse DAE (below 1% density) can achieve more than 90.0% accuracy on MNIST while having about 150 times fewer parameters than FC-DAE.

C.5 Feature Selection on a Large Dataset

In this appendix, we evaluate the performance of the methods on a very large dataset, in terms of both number of samples and dimensions.

In this experiment, first, we generate two artificial datasets with high number of samples and features. The choice of an artificial dataset was made to easily control the number of relevant features of the dataset, as in most of the real-world datasets the number of informative features are not clear. These datasets

are generated using *sklearn*[1] library tools, *make_classification* function, which generates datasets with a desired number of features and samples. This function allows us to adjust the number of informative, redundant, and non-informative features. Table C.1 shows the characteristics of the two artificially generated datasets. We generated 2 datasets with 40000 samples and 8000 features. However, the number of informative and redundant features are different in these datasets. Artificial2 dataset is much noisier than Artificial1; therefore, finding relevant features of Artificial2 is more difficult compared to finding them on the Artificial1 dataset.

Table C.1: Characteristics of the two artificially generated datasets. The classification and clustering accuracy have been obtained using all the features.

Dataset	Samples	Features	Informative Features	Redundant Features	Classes	Classification Accuracy (%)	Clustering Accuracy (%)
Artificial1	40000	8000	500	3000	5	59.8	30.6
Artificial2	40000	8000	1000	0	5	26.6	22.7

After generating the datasets, we evaluate feature selection performance of the methods considered in the manuscript, and compare the results with QuickSelection. The hyperparameters used in this experiment are similar to the ones used in Section 4.4.1, except for hidden neurons and the sparsity level. The number of hidden neurons for autoencoder-based methods has been set to 2000, and the hyperparameter of QuickSelection, ϵ, has been adjusted to 40. The number of selected features (K) is 1000. The number of training epochs for the autoencoder-based methods is 100. However, since the QuickSelection did not converge in 100 epochs on the Artificial2 dataset, we continued the training until epoch 200. The results of this experiment are presented in Table C.2.

As can be seen in Table C.2, QuickSelection$_{100}$ outperforms all the other methods in terms of classification accuracy on both datasets. It can also outperform the other methods in terms of clustering accuracy on the Artificial2 dataset. As mentioned earlier, QuickSelection achieves a higher accuracy on the Artificial2 dataset when it is trained for more than 100 epochs. However, since for all the other methods we use 100 epochs, we only consider the results of QuickSelection$_{100}$ to have a fair comparison (it should be noted that increasing the number of training epochs did not improve the results of the other methods).

[1] https://scikit-learn.org/

Table C.2: Feature selection results on two artificially generated datasets ($K = 1000$)

Method	Artificial1 Dataset		Artificial2 Dataset		Number of Parameters
	Classification Accuracy (%)	Clustering Accuracy (%)	Classification Accuracy (%)	Clustering Accuracy (%)	
Lap_score	49.4	24.4	22.0	21.3	-
MCFS	68.2	**29.3**	24.6	21.7	-
CAE	23.4	20.6	21.1	20.4	$\sim 26\times10^6$
AEFS	34.7	23.3	22.8	21.4	$\sim 32\times10^6$
FCAE	43.8	24.3	22.9	21.5	$\sim 32\times10^6$
QS$_{10}$	68.1	25.7	24.8	21.21	$\sim 0.8\times10^6$
QS$_{100}$	**68.4**	24.8	**34.5**	**24.6**	$\sim 0.8\times10^6$
QS$_{200}$	-	-	**39.7**	**29.6**	$\sim 0.8\times10^6$

On noisy and very large datasets, CAE, AEFS, and FCAE have a poor performance in feature selection. In addition, they have around 30 times more parameters than QuickSelection. CAE has the lowest accuracy among these methods; this method is very sensitive to noise. Lap_score and MCFS have a poor performance on the Artificial2 dataset that is noisier than Artificial1. On the Artificial1 dataset, MCFS achieves the highest clustering accuracy. However, the memory requirement of MCFS and Lap_score is noticeably large. On this dataset, they need about $26GB$ of RAM. However, QuickSelection needs only about $8GB$ memory. In summary, QuickSelection$_{100}$ has a decent performance on these large datasets, while having the lowest number of parameters.

C.6 Sparse Training Algorithm Analysis

In this appendix, we aim to analyze the effect of the SET training procedure on the performance of QuickSelection.

We perform QuickSelection using another algorithm to obtain and train the sparse network, and then, compare the result with the original QuickSelection. We derive the sparse denoising autoencoder using the lottery ticket hypothesis algorithm [FC19], as follows. The lottery ticket hypothesis (LTH), first, starts with training a dense network. After that, it derives the topology of the sparse network by pruning the unimportant weights of the trained dense network. Then, using both the sparse topology and the initial weight values of the connections

in the dense training phase, the network is retrained. On the final obtained sparse model, we apply QuickSelection principles to select the most informative features.

In this experiment, the structure, sparsity level, and other hyperparameters are similar to the settings described in Section 4.4.1; we use a simple autoencoder with one hidden layer containing 1000 hidden neurons, trained for 100 epochs. The results of feature selection ($K = 50$) are available in Tables C.3 and C.4. We refer to the feature selection performed using QuickSelection principles and the Sparse DAE obtained with LTH as QS_{100}^{LTH}. We use QS_{100} for the QuickSelection that is done using the Sparse DAE obtained with SET.

As can be observed in Tables C.3 and C.4, in most of the cases QS_{100} outperforms QS_{100}^{LTH}. We believe that optimizing the sparse topology and the weights,

Table C.3: Clustering accuracy (%) using 50 selected features (except Madelon for which we select 20 features).

Method	Coil20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
QS_{100}	**60.2±2.0**	**35.1±2.7**	**54.6±4.5**	**58.2±1.5**	**48.3±2.4**	51.8±0.8	**59.5±1.8**	**52.5±1.1**
QS_{100}^{LTH}	58.8±3.3	31.2±2.4	50.2±6.3	50.8±0.5	37.5±4.0	**54.6±2.7**	54.6±3.7	50.8±0.6

Table C.4: Classification accuracy (%) using 50 selected features (except Madelon for which we select 20 features).

Method	Coil20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
QS_{100}	**99.7±0.3**	**89.0±1.3**	**90.2±1.2**	**90.3±0.7**	**93.5±0.5**	**75.7±3.9**	**73.3±3.3**	58.0±2.9
QS_{100}^{LTH}	99.6±0.6	84.5±3.9	86.3±6.3	53.0±7.2	82.6±2.4	74.2±2.7	71.3±4.2	**59.5±5.9**

Table C.5: Number of parameters of QS_{100} and QS_{100}^{LTH} (divided by 10^6).

Method	Coil20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
QS_{100}	**0.054**	**0.043**	**0.042**	**0.040**	**0.048**	**0.566**	**1.3**	**0.115**
QS_{100}^{LTH}	2.054	1.243	1.142	1.040	1.548	40.566	99.3	6.715

simultaneously, results in feature strength that are more meaningful for the feature selection. We discussed neuron strength in more detail in Section 4.5.3. In addition, due to having an extra phase of dense training, the computational resource requirements of LTH are much higher than the ones of SET. To clarify this aspect, we present a comparison for the number of parameters between these two methods. The results can be found in Table C.5. The much higher number of parameters in QS_{100}^{LTH} in comparison with the number of parameters in QS_{100} is given by the dense training phase of LTH.

C.7 Performance Evaluation using Random Forest Classifier

In this appendix, we validate the classification accuracy results using another classifier. We repeat the experiment from Section 4.4.2 in the manuscript; however, we measure the accuracy of selecting 50 features (for Madelon, we select 20 features) using the RandomForest classifier [LW+02] instead of the ExtraTrees classifier. The results are presented in Table C.6.

Table C.6: Classification accuracy (%) using 50 selected features (except Madelon for which we select 20 features). On each dataset, the bold entry is the best performer, and the italic one is the second-best performer. The classifier used for evaluation is the random forest classifier.

Method	COIL-20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
MCFS	**99.5±0.3**	79.9±0.4	88.5±0.4	81.9±0.7	89.2±0.0	76.3±3.7	69.4±3.9	56.5±0.16
LS	88.9±0.8	83.4±0.2	86.4±0.3	**88.9±0.6**	20.7±0.1	67.9±3.1	71.1±2.8	50.13±0
CAE	*99.3±0.6*	*89.0±0.7*	**89.8±1.0**	*84.2±0.9*	**95.2±0.2**	*76.7±4.7*	**76.6±3.8**	*61.6±2.3*
AEFS	92.4±2.3	84.9±1.7	87.8±1.1	59.6±4.0	87.6±0.8	71.1±6.2	67.2±4.8	57.7±2.2
FCAE	99.0±0.6	85.8±5.2	83.6±2.6	62.7±13.1	69.6±2.9	74.2±2.6	68.9±4.0	58.8±2.5
QS_{10}	98.5±0.9	87.0±0.7	87.6±0.5	81.5±3.8	*93.6±0.6*	75.1±2.3	68.1±4.6	60.0±3.7
QS_{100}	**99.5±0.3**	**89.1±1.3**	*89.0±1.2*	**88.9±0.7**	93.2±0.5	**78.5±3.5**	*73.3±3.1*	**67.9±3.8**

As can be seen in Table C.6, $QuickSelection_{100}$ is the best performer in 5 out of 8 cases. By comparing the results with Table 4.3 which demonstrates the classification accuracy measured by the ExtraTrees classifier, it is clear that there have been subtle changes in the accuracy values. This has resulted in some

changes in the ranking of the methods in terms of performance, as in several cases, the performance of the methods is very close. The reason behind choosing ExtraTrees classifier in the experiment was due to the low computational cost. However, as discussed in Chapter 4, to perform an extensive evaluation, we have also measured the performance using clustering accuracy. Overall, by looking into the results of the three approaches to compute accuracy, it is clear that QuickSelection is a performant feature selection method in terms of the quality of the selected features.

C.7.1 SET Hyperparameters

As explained in Chapter 4, ζ and ϵ are the hyperparameters of the SET algorithm which control the number of connections to remove/add for each topology change and the sparsity level, respectively. The corresponding density level of each ϵ value for each dataset can be observed in Table C.7.

Table C.7: ϵ values and their corresponding density level.

	Density [%]							
ϵ	COIL-20	Isolet	HAR	Madelon	MNIST	SMK	GLA	PCMAC
2	0.39	0.52	0.39	0.59	0.45	0.20	0.2	0.26
5	0.98	1.30	0.98	1.48	1.13	0.53	0.51	0.65
10	1.95	2.58	1.95	2.95	2.25	1.04	1.02	1.13
13	2.53	3.35	2.53	3.82	2.91	1.35	1.32	1.69
20	3.87	5.10	3.87	5.82	4.45	2.07	2.04	2.6
25	4.87	6.45	4.87	7.37	5.63	2.65	2.55	3.26

To illustrate the effect of the hyperparameters ζ and ϵ, we perform a grid search within a small set of values on all of the datasets. The obtained results can be found in Tables C.8 and C.9. As we increase the ϵ value, the number of connections in our model increases, and therefore, the computation time will increase. So, we prefer using small values for this parameter. Additionally, for a large value of ϵ, in some cases the model is not able to converge in 100 epochs; for example, on the MNIST dataset, we can observe that for an ϵ value of 25, the model has lower performance in terms of clustering and classification accuracy.

It can be observed that $\zeta = 0.2$ and $\epsilon = 13$ (as chosen for the experiments performed in Chapter 4) lead to a decent performance on all datasets. For these values, QuickSelection is able to achieve high clustering and classification

accuracy.

Overall, although searching for the best pair of ζ and ϵ will improve the performance, QuickSelection is not extremely sensitive to these values. As can be seen in Tables C.8 and C.9, for all values of these hyperparameters QuickSelection has a reasonable performance. Even with $\epsilon = 2$ which leads to a very sparse model, QuickSelection has decent performance, and in some cases better than a denser network.

Table C.8: Hyper-parameter selection for QuickSelection$_{10}$. Each entry of a table contains clustering accuracy and classification accuracy in percentages (%), respectively.

(a) Coil20

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	61.4±2.3, 98.6±1.5	59.7±1.8, 96.9±1.6	61.7±2.8, 98.3±0.8	60.3±2.0, 98.6±1.7	61.9±0.9, 99.5±0.3	60.8±2.4, 99.7±0.2
0.2	59.4±3.8, 97.9±1.4	59.3±0.9, 98.5±1.2	60.1±2.6, 98.9±0.6	59.5±2.1, 98.8±0.6	63.6±2.5, 99.7±0.3	61.0±3.0, 99.7±0.4
0.3	61.3±2.1, 98.7±1.6	59.5±1.9, 96.7±0.6	60.3±0.9, 99.1±0.6	60.1±1.9, 98.2±1.0	59.9±2.6, 98.7±0.9	60.8±1.9, 99.4±0.4
0.4	60.5±3.2, 97.5±1.3	59.0±2.4, 96.8±1.8	57.2±1.5, 97.7±1.7	62.0±2.5, 99.2±0.3	62.1±2.5, 99.7±0.3	63.8±1.5, 99.4±0.6
0.5	61.2±2.3, 98.0±0.9	58.4±2.6, 97.8±1.2	58.9±2.6, 97.3±1.3	60.6±1.6, 98.2±1.6	59.1±2.9, 99.0±0.6	62.8±1.6, 99.0±1.0

(b) Isolet

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	28.6±2.1, 82.5±3.7	31.7±2.4, 84.9±2.4	31.2±2.2, 87.4±1.7	29.4±1.5, 88.4±1.3	32.4±1.8, 86.9±1.5	31.7±1.0, 86.9±0.7
0.2	26.1±2.2, 81.5±1.8	30.2±2.4, 86.2±2.8	30.9±2.3, 88.2±0.6	32.5±2.8, 86.9±1.1	32.4±1.6, 87.8±1.6	33.4±2.1, 87.5±1.0
0.3	27.9±2.0, 83.0±3.6	30.6±3.3, 87.2±2.1	31.9±2.5, 87.3±1.1	32.9±1.1, 87.7±1.1	32.2±1.8, 87.3±0.9	32.3±2.4, 88.1±1.7
0.4	26.9±2.0, 82.5±2.5	30.0±0.8, 86.4±3.3	31.5±2.3, 86.2±2.6	29.0±1.4, 85.4±1.6	34.5±2.9, 86.5±2.3	31.9±3.4, 87.1±1.1
0.5	26.9±1.9, 81.8±1.7	30.7±1.9, 84.9±3.1	30.7±2.3, 86.7±2.3	31.2±3.5, 86.9±1.7	33.1±1.5, 87.4±0.8	33.8±2.1, 87.1±0.6

(c) HAR

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	43.0±2.7, 83.2±1.6	56.8±1.0, 88.6±1.5	55.8±3.8, 88.4±0.6	56.0±2.6, 87.7±0.8	56.6±4.9, 90.1±1.9	56.4±4.7, 88.4±1.3
0.2	43.8±2.1, 82.7±2.6	56.5±0.9, 89.2±2.2	58.8±1.1, 88.4±0.5	56.0±2.6, 88.8±0.7	54.5±4.1, 88.9±1.3	55.0±2.8, 89.7±0.8
0.3	43.0±2.1, 84.5±2.2	55.7±1.7, 89.6±1.4	59.2±1.0, 88.4±1.1	54.7±5.2, 88.4±1.5	56.3±2.4, 89.1±1.8	57.7±2.1, 89.3±1.7
0.4	42.4±3.7, 82.9±1.4	55.5±1.4, 89.8±1.5	56.9±1.8, 88.7±0.6	59.1±0.7, 89.4±0.6	56.7±3.2, 89.7±1.2	56.9±2.2, 89.1±1.8
0.5	45.9±7.0, 84.0±3.1	52.2±5.2, 89.9±0.7	57.9±1.9, 89.2±0.6	59.5±4.3, 89.7±1.1	55.3±2.7, 89.5±0.6	52.6±2.6, 89.9±0.8

(d) Madelon

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	53.5±1.3, 55.3±1.9	54.0±4.0, 62.1±7.9	56.7±3.8, 78.5±7.0	56.6±2.5, 86.2±2.1	58.1±2.5, 87.6±1.9	56.5±3.9, 89.4±1.1
0.2	52.9±2.8, 61.4±5.6	57.0±3.2, 68.6±4.2	57.6±2.9, 83.6±3.5	57.5±3.8, 86.6±3.6	55.5±3.7, 88.3±0.7	59.1±0.9, 86.0±1.5
0.3	53.4±2.5, 58.7±10.3	56.4±3.5, 67.1±9.1	53.9±3.1, 81.4±5.6	54.4±3.3, 86.4±2.0	58.2±2.7, 88.3±1.8	55.5±2.6, 88.5±0.8
0.4	53.8±4.1, 55.3±6.3	55.9±4.3, 62.6±6.4	54.4±2.4, 80.1±3.6	53.6±2.4, 84.9±3.2	56.8±3.0, 89.7±0.8	58.1±1.8, 86.4±2.8
0.5	55.2±3.8, 56.0±3.9	52.2±2.3, 61.3±9.0	56.0±2.4, 81.7±3.9	57.9±2.8, 84.2±4.6	57.8±2.3, 86.9±0.6	57.2±2.5, 89.0±2.1

(e) MNIST

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	42.6±3.5, 91.4±0.5	41.2±1.8, 93.2±0.4	46.8±4.4, 94.3±0.2	46.9±1.3, 93.9±0.3	43.3±1.7, 93.5±0.4	39.5±1.7, 92.4±1.2
0.2	43.8±2.3, 92.5±0.6	43.9±1.4, 93.4±0.6	43.0±3.2, 93.6±0.4	45.4±3.9, 93.8±0.6	38.5±2.9, 92.7±0.6	37.6±3.4, 91.2±1.1
0.3	42.1±1.8, 91.9±0.5	45.4±2.4, 94.2±0.4	47.4±1.2, 94.0±0.1	45.5±3.3, 94.0±0.4	39.7±1.9, 92.8±0.7	33.9±3.3, 88.3±2.1
0.4	41.9±2.0, 92.9±0.7	46.9±2.6, 93.7±0.4	46.2±0.9, 94.1±0.3	43.4±1.5, 93.7±0.3	37.0±3.6, 90.9±1.5	27.9±2.2, 81.7±3.1
0.5	43.3±3.6, 92.3±0.5	45.9±4.8, 93.8±0.6	45.2±2.9, 93.8±0.4	42.8±2.7, 93.8±0.7	39.7±2.8, 91.3±0.6	28.0±2.5, 77.7±6.1

(f) SMK

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	52.4±1.3, 72.1±7.0	53.8±3.0, 79.5±2.0	52.7±1.6, 76.8±4.5	56.0±1.7, 73.7±4.1	55.0±2.3, 74.2±7.3	53.7±2.4, 76.3±4.4
0.2	54.1±1.7, 73.7±4.7	53.5±2.7, 74.2±8.4	55.3±2.6, 75.3±4.9	54.0±3.1, 76.9±4.6	52.9±1.2, 81.6±5.0	54.5±3.0, 76.8±6.3
0.3	56.9±2.7, 76.8±6.1	54.7±1.3, 75.3±5.2	53.9±2.4, 74.7±4.9	53.9±2.3, 74.2±4.5	54.5±0.7, 76.3±3.7	54.8±2.9, 75.8±3.9
0.4	55.4±3.7, 74.7±2.1	55.5±1.7, 74.2±2.6	53.1±1.8, 72.6±4.9	52.8±1.6, 74.7±4.6	53.4±2.9, 72.6±4.3	53.1±2.5, 72.6±4.3
0.5	53.3±1.3, 77.4±5.4	55.2±3.0, 76.3±3.7	53.6±2.5, 76.3±6.0	52.5±1.5, 78.9±4.4	52.3±1.0, 77.4±7.7	51.9±1.5, 77.9±2.7

(g) GLA

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	54.1±2.8, 66.7±5.0	54.7±3.5, 67.2±5.9	55.0±4.6, 66.7±1.8	54.5±1.5, 67.8±5.7	56.6±4.0, 75.0±6.3	55.9±3.2, 68.9±5.4
0.2	50.2±3.5, 67.8±3.8	53.4±3.3, 67.2±6.2	56.6±2.5, 70.0±4.4	53.6±4.7, 69.4±3.0	56.7±2.2, 68.3±2.8	52.6±1.5, 68.9±1.1
0.3	56.2±3.5, 68.9±4.8	53.3±4.8, 68.3±3.8	54.4±2.4, 67.8±2.8	57.8±4.3, 70.0±3.2	56.1±1.9, 70.6±3.8	56.0±3.0, 71.1±4.5
0.4	55.6±3.5, 68.9±2.1	54.2±1.5, 68.3±4.5	57.5±3.1, 68.3±2.2	56.9±1.1, 70.6±2.8	55.7±3.6, 68.3±4.5	55.4±2.4, 68.9±6.9
0.5	54.9±2.6, 68.9±4.1	54.0±2.5, 66.1±3.7	54.8±2.4, 71.1±4.5	54.5±5.1, 67.2±6.4	56.5±5.6, 71.1±1.4	55.8±2.0, 65.6±3.8

(h) PCMAC

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	51.0±0.5, 61.1±4.2	51.0±0.6, 57.0±2.0	51.1±1.1, 59.3±3.4	51.4±0.5, 56.6±3.0	50.9±0.2, 55.5±3.5	51.3±0.5, 59.4±2.2
0.2	50.5±0.4, 61.3±6.1	50.8±0.5, 57.0±3.5	50.7±0.4, 55.8±2.1	50.9±0.5, 58.9±4.4	51.0±0.2, 59.2±4.0	51.0±0.6, 57.8±2.1
0.3	51.3±1.0, 58.7±2.9	50.9±0.3, 57.4±1.1	51.0±0.4, 57.0±2.0	51.2±0.5, 59.2±3.2	50.7±0.3, 58.2±1.9	51.1±0.6, 58.3±1.9
0.4	50.7±0.3, 58.1±2.4	51.3±0.4, 55.7±2.8	50.9±0.5, 55.2±1.0	51.1±0.3, 58.1±2.5	51.1±0.2, 57.9±3.7	51.6±0.9, 55.4±2.2
0.5	51.1±0.5, 57.4±2.4	51.1±0.4, 57.0±1.6	51.2±0.9, 58.1±3.0	51.0±0.6, 56.4±1.4	50.9±0.3, 55.8±1.9	51.6±0.9, 58.0±2.4

Table C.9: Hyper-parameter selection for QuickSelection$_{100}$. Each entry of each table contains clustering accuracy and classification accuracy in percentages (%), respectively.

(a) Coil20

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	63.2±0.7, 99.7±0.2	62.8±1.1, 99.4±0.7	60.2±3.5, 99.2±0.4	61.8±1.5, 99.7±0.5	56.0±2.3, 98.8±1.0	53.4±1.7, 98.8±0.5
0.2	61.3±0.9, 99.1±0.7	62.1±3.2, 99.7±0.1	61.7±2.3, 99.6±0.4	60.2±2.0, 99.7±0.3	56.9±1.8, 99.1±0.6	53.9±1.5, 98.9±0.7
0.3	62.1±1.5, 98.5±0.8	62.0±2.6, 99.4±0.7	60.0±1.8, 99.5±0.2	60.2±2.5, 99.3±0.2	55.0±1.6, 98.8±0.9	53.8±1.7, 98.3±0.8
0.4	58.9±1.3, 98.3±0.6	62.9±1.0, 99.7±0.3	62.0±3.0, 99.5±0.5	62.3±1.4, 99.7±0.4	57.8±2.5, 99.2±0.2	57.2±2.2, 99.0±0.7
0.5	58.1±1.9, 97.1±1.7	59.9±1.5, 99.4±0.4	63.2±2.6, 99.0±0.8	64.2±1.3, 99.6±0.3	59.2±2.9, 98.8±1.1	58.0±1.4, 99.1±1.0

(b) Isolet

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	29.4±2.2, 87.1±1.1	29.7±1.5, 84.8±3.2	28.3±2.7, 83.4±4.2	33.2±3.0, 89.3±1.8	37.7±1.9, 87.5±1.8	36.2±2.4, 88.3±1.2
0.2	29.4±2.2, 85.9±2.1	29.6±2.7, 86.0±1.8	31.5±2.0, 85.5±3.7	35.1±2.7, 89.0±1.3	35.5±2.5, 87.5±2.2	38.9±1.7, 87.5±0.4
0.3	30.3±2.2, 85.7±3.1	30.2±1.8, 84.2±3.8	30.0±2.6, 84.5±1.8	33.5±2.3, 87.6±1.8	35.7±3.0, 87.1±2.5	38.1±1.7, 87.4±1.7
0.4	31.1±3.3, 85.9±3.8	29.5±2.5, 86.1±3.2	30.4±3.4, 83.7±3.5	29.6±1.3, 85.4±0.8	33.1±3.6, 87.6±2.7	35.4±1.5, 87.9±1.2
0.5	30.4±2.5, 88.0±2.1	29.5±1.8, 86.2±2.7	31.5±2.7, 86.4±1.9	31.4±2.1, 86.2±1.9	33.3±2.0, 86.4±2.1	35.7±2.0, 89.5±0.6

(c) HAR

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	52.7±5.2, 88.4±2.3	57.0±1.4, 89.5±1.2	56.1±1.9, 88.8±1.2	54.2±4.1, 88.5±2.1	56.0±2.0, 89.2±2.5	55.2±3.5, 87.5±2.1
0.2	48.9±4.0, 85.8±1.7	57.4±0.4, 90.0±0.6	52.1±4.5, 88.6±2.7	54.6±4.5, 90.2±1.2	54.2±1.8, 89.4±0.7	53.9±3.0, 89.2±1.9
0.3	50.9±6.7, 88.5±2.8	56.3±6.4, 89.5±1.2	54.2±3.6, 90.8±1.5	53.6±4.4, 90.5±3.2	52.0±6.3, 88.0±1.4	51.1±3.1, 89.4±1.1
0.4	48.9±6.1, 88.7±2.6	55.7±6.6, 90.5±1.4	54.1±4.1, 91.1±0.3	50.8±4.1, 89.2±1.3	49.7±3.2, 90.2±1.1	54.1±4.4, 89.0±1.9
0.5	46.0±5.7, 87.3±3.2	55.5±8.1, 90.5±1.6	52.8±5.3, 90.2±1.3	55.4±0.7, 90.4±1.0	50.4±4.3, 89.4±0.4	49.6±5.3, 88.3±1.1

(d) Madelon

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	53.7±3.3, 75.1±6.8	58.5±2.8, 86.1±3.5	57.1±2.2, 90.3±1.0	56.7±2.2, 89.9±1.2	58.4±0.5, 90.3±0.8	58.3±0.5, 89.8±1.1
0.2	54.3±2.4, 81.2±4.7	55.6±2.5, 88.2±2.1	57.1±2.6, 89.6±0.9	58.2±1.5, 90.3±0.7	58.1±0.1, 90.3±1.3	58.1±0.0, 90.8±0.5
0.3	53.1±2.6, 82.0±4.5	60.1±0.8, 87.5±1.3	57.6±2.0, 89.6±1.2	57.6±1.5, 89.4±1.3	58.1±0.0, 90.9±0.4	58.3±0.5, 89.5±1.5
0.4	55.0±2.8, 78.6±8.3	58.4±2.9, 87.0±4.6	55.8±2.2, 90.6±0.6	58.4±0.7, 90.1±0.9	57.4±1.6, 90.3±0.7	58.1±0.0, 90.9±1.2
0.5	55.6±3.2, 74.3±3.3	57.1±3.2, 87.1±2.4	57.1±3.5, 90.0±0.6	58.9±0.4, 90.3±0.3	58.5±0.6, 89.3±0.7	58.5±0.4, 89.4±1.3

(e) MNIST

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	44.1±2.2, 92.8±1.0	46.3±3.4, 94.0±0.4	48.3±1.7, 94.0±0.6	44.3±2.7, 93.8±0.5	43.5±3.4, 92.8±0.5	31.3±4.3, 85.2±4.9
0.2	43.3±5.1, 93.5±1.2	44.3±1.4, 93.7±0.3	47.1±3.1, 93.7±0.6	48.3±2.4, 93.5±0.5	37.7±1.3, 91.3±0.4	33.7±3.5, 87.8±1.5
0.3	45.1±2.8, 93.2±0.3	48.4±4.5, 93.7±0.8	46.0±4.6, 93.4±0.3	44.9±4.4, 93.8±0.4	35.8±3.0, 91.6±0.9	38.1±1.3, 90.8±0.6
0.4	45.1±2.4, 93.4±0.3	45.4±2.3, 94.2±0.3	45.0±2.1, 93.4±0.5	40.1±4.0, 92.4±0.7	41.5±4.9, 91.8±1.3	32.5±2.7, 87.7±3.7
0.5	45.7±2.6, 93.8±0.7	44.1±2.6, 93.7±0.6	43.8±2.6, 93.8±0.5	43.2±1.8, 92.6±0.7	36.6±4.5, 91.5±1.0	36.0±2.7, 88.5±1.4

(f) SMK

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	53.1±1.3, 72.6±5.9	52.6±1.6, 76.3±2.4	51.6±0.9, 76.8±4.8	53.5±1.6, 75.8±2.6	54.6±3.2, 72.6±2.1	51.4±0.9, 76.8±5.4
0.2	53.3±2.3, 74.2±5.4	53.0±1.5, 76.8±3.1	51.1±0.6, 78.4±5.9	51.8±0.8, 75.7±3.9	51.3±0.6, 78.4±3.5	51.9±1.6, 78.4±6.5
0.3	53.3±1.7, 74.2±4.5	50.7±0.3, 76.3±6.5	51.6±0.9, 76.8±6.1	50.9±0.5, 74.7±3.6	51.2±0.6, 77.4±2.7	51.4±0.7, 77.4±5.4
0.4	52.4±2.6, 78.4±4.5	51.6±0.7, 78.4±2.6	50.9±0.4, 75.8±5.4	51.9±0.8, 75.3±7.2	51.1±0.5, 76.8±4.5	50.8±0.3, 76.3±2.4
0.5	53.0±1.3, 76.8±6.1	52.1±1.1, 74.7±2.7	51.9±0.8, 74.2±3.5	50.7±0.4, 75.8±3.9	50.3±0.0, 78.9±4.1	51.1±0.5, 81.0±4.2

(g) GLA

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	57.6±2.6, 68.9±5.9	57.1±1.9, 67.2±1.1	57.4±2.8, 72.2±3.9	57.7±2.9, 68.9±3.2	57.4±2.9, 73.3±4.8	59.2±2.7, 71.7±4.1
0.2	57.0±3.4, 64.4±3.2	60.8±3.8, 71.1±3.3	58.7±3.5, 67.8±6.0	59.5±1.8, 73.3±3.3	58.6±2.0, 72.8±2.1	55.6±1.3, 70.6±7.2
0.3	57.7±3.5, 73.9±3.3	58.3±4.1, 67.2±3.2	54.8±0.9, 72.2±3.5	58.0±4.3, 67.8±3.8	56.4±3.5, 68.3±4.2	57.3±2.8, 66.7±2.5
0.4	56.1±2.6, 71.1±3.3	57.9±2.9, 67.2±4.8	54.4±2.5, 67.2±3.2	59.0±4.0, 69.4±4.6	56.9±2.3, 69.4±2.5	59.9±3.6, 69.4±4.6
0.5	55.2±2.2, 67.2±6.4	56.0±1.7, 63.9±1.8	58.0±2.2, 68.3±6.0	59.0±3.1, 70.0±5.4	59.5±3.2, 71.1±6.2	53.6±1.7, 68.3±4.2

(h) PCMAC

	ϵ					
ζ	**2**	**5**	**10**	**13**	**20**	**25**
0.1	50.6±0.3, 58.1±3.8	50.8±0.4, 57.4±3.1	51.4±1.2, 58.5±2.3	51.0±0.4, 59.2±3.2	50.8±0.2, 59.2±3.1	52.6±1.0, 58.8±3.4
0.2	50.7±0.4, 59.4±2.9	50.7±0.5, 60.6±3.4	52.1±1.7, 57.2±3.4	52.5±1.1, 58.0±2.9	53.1±0.0, 58.6±2.6	53.1±0.0, 60.1±2.0
0.3	51.5±0.9, 57.2±2.9	51.4±0.9, 56.0±2.2	51.7±1.2, 58.1±0.9	52.2±1.1, 56.5±1.7	53.1±0.0, 59.5±2.4	53.1±0.0, 57.3±4.1
0.4	50.9±0.4, 59.8±6.7	51.3±0.9, 56.3±4.1	52.0±1.3, 57.3±3.0	53.1±0.0, 56.7±2.2	53.1±0.0, 56.6±2.0	53.1±0.0, 57.6±2.0
0.5	50.7±0.2, 56.9±0.5	51.3±0.9, 57.1±2.1	52.6±0.9, 59.6±1.9	53.1±0.0, 57.7±1.8	53.1±0.0, 56.8±3.4	53.1±0.0, 59.8±1.6

Appendix D

Additional Experiments and Analysis for Chapter 5

D.1 Ablation Study: Gradient vs Random Policy for Weight and Neuron Selection

This Appendix discusses the effect of gradient-based weights and neuron selection in NeuroFS by performing an ablation study. We use random growth instead of the gradient to measure the importance of weights and neurons. We call this method NeuroFS[w/oGradient]. The settings of this experiment is similar to Section 5.4.2. The results are presented in Figure D.1.

In Figure D.1, NeuroFS outperforms NeuroFS[w/oGradient] in most cases. While the results of these methods are relatively close on some datasets, on the Coil-20, SMK, GLA-BRA-180, and Arcene datasets, there is a large gap between the results. It can be concluded that NeuroFS performs more stable than NeuroFS[w/oGradient]. While random growth of weights and neurons might lead to better results in some cases, it can not ensure a stable performance across different datasets.

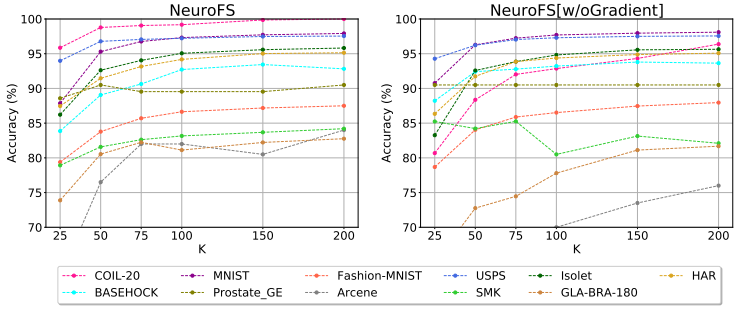


Figure D.1: Gradient (left) vs. random (right) weight and neuron growth policy comparison.

D.2 Comparison with HSICLasso-based Feature Selection Methods

In this section, we compare NeuroFS with two HSIC-based feature selection methods. We select two algorithms including, HSICLasso[1] [YJS+14] and HSI-

[1] *https://github.com/riken-aip/pyHSICLasso*

CLassoVI[2] [KKOI22] and used the default hyperparameters used in the corresponding code repositories. The results are presented in Table D.1. As can be seen in this table, NeuroFS outperforms these methods in seven cases while performing very close to the best performer in the other cases (less than a 1% difference in accuracy).

Table D.1: Supervised feature selection comparison (average classification accuracy for K values in [25, 50, 75, 100, 150, 200] (%)) with HSICLasso-based methods. Empty entries show that the corresponding experiments ran into error. Bold and italic fonts indicate the best and second-best performer, respectively.

	Low-dimensional Datasets						High-dimensional Datasets				
Method	**COIL-20**	**MNIST**	**F-MNIST**	**USPS**	**Isolet**	**HAR**	**BASEHOCK**	**Prostate_GE**	**Arcene**	**SMK**	**GLA**
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	*98.79±0.22*	**95.48±0.34**	**85.03±0.15**	**96.68±0.16**	**93.22±0.11**	**92.74±0.23**	**90.42±0.80**	*89.70±0.72*	**78.00±1.78**	*82.36±0.98*	*80.46±0.99*
HSICLasso	**99.32±0.00**	*93.80±0.00*	*82.73±0.00*	96.03±0.00	*91.07±0.00*	*92.68±0.00*	*88.62±0.00*	**90.50±0.00**	*77.50±0.00*	70.60±0.00	**80.55±0.00**
HSICLassoVI	92.83±0.00	-	75.72±0.00	*96.15±0.00*	75.43±0.00	89.10±0.00	-	-	-	**83.33±0.00**	-

D.3 Feature Selection Comparison using Different Classifiers for Evaluation

To show that the evaluation results are not biased by the chosen classifier, we measure the classification accuracy by using two other widely-used classifiers including KNN[3] and ExtraTrees[4]. The classification accuracy results are presented in Table D.2 for $K = 50$. As can be seen in this table, the overall performance of all methods is consistent across different classifiers in most cases.

D.4 Comparison to RigL

In this section, we compare NeuroFS with RigL [EGM+20], which is a DST method mainly designed for classification; it uses gradient for weight regrowth when updating the sparse connectivity in the DST framework. To adapt RigL to perform feature selection, while trying to keep a fair comparison, we take the

[2] *https://github.com/nttcom/HSICLassoVI*

[3] *https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html*

[4] *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html*

Table D.2: Supervised feature selection comparison (classification accuracy for $K = 50$ (%)) using different classifiers. Empty entries show that the corresponding experiments exceeded the time limit (12 hours). Bold and italic fonts indicate the best and second-best performer, respectively.

	Low-dimensional Datasets						High-dimensional Datasets				
Method	**COIL-20**	**MNIST**	**F-MNIST**	**USPS**	**Isolet**	**HAR**	**BASEHOCK**	**Prostate_GE**	**Arcene**	**SMK**	**GLA**
					SVM						
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	98.78±0.29	**95.30±0.41**	**83.78±0.64**	**96.78±0.17**	**92.62±0.40**	91.46±0.72	*89.06±2.46*	**90.50±0.00**	*76.50±2.55*	*81.58±1.68*	**80.54±4.96**
LassoNet	97.16±1.06	*94.46±0.21*	*82.58±0.10*	95.94±0.15	84.90±0.22	*93.74±0.39*	87.18±0.58	*88.58±2.35*	71.00±2.00	80.52±2.69	*74.46±4.78*
STG	**99.32±0.40**	93.20±0.62	82.36±0.52	*96.62±0.34*	85.82±2.83	91.22±1.23	85.12±1.86	84.78±3.55	71.00±2.55	80.25±2.95	70.00±4.08
QS	96.52±1.53	93.62±0.49	80.82±0.51	95.52±0.27	89.78±1.80	91.96±1.04	87.22±1.22	76.20±7.53	74.38±4.80	80.90±2.20	72.20±2.80
Fisher_score	74.00±0.00	81.90±0.00	67.80±0.00	91.00±0.00	67.40±0.00	79.80±0.00	**90.20±0.00**	**90.50±0.00**	67.50±0.00	73.70±0.00	63.90±0.00
CIFE	59.40±0.00	89.30±0.00	66.90±0.00	61.30±0.00	59.80±0.00	84.20±0.00	77.40±0.00	47.60±0.00	52.50±0.00	**81.60±0.00**	58.30±0.00
ICAP	*99.30±0.00*	89.00±0.00	59.50±0.00	95.20±0.00	75.10±0.00	88.70±0.00	**90.20±0.00**	57.10±0.00	70.00±0.00	73.70±0.00	72.20±0.00
RFS	95.80±0.00	-	-	95.80±0.00	*91.50±0.00*	**94.00±0.00**	85.00±0.00	**90.50±0.00**	**77.50±0.00**	-	-
					KNN						
Baseline	100.0	96.91	84.96	97.37	88.14	87.85	78.7	76.19	92.5	73.68	69.44
NeuroFS	*99.80±0.28*	**91.64±0.57**	**80.12±0.87**	**96.18±0.49**	*85.96±1.53*	84.64±1.77	87.14±2.69	*85.86±4.67*	74.00±5.15	**78.97±4.55**	64.42±5.38
LassoNet	98.84±0.20	*91.38±0.36*	*79.30±0.20*	*95.70±0.26*	79.22±0.47	*88.70±0.57*	88.96±1.20	82.86±3.80	67.50±7.75	*74.74±6.34*	**68.90±4.07**
STG	**99.94±0.12**	87.16±0.64	77.65±0.48	95.14±0.45	83.16±3.42	87.86±0.39	81.10±1.93	81.00±0.00	*75.00±5.24*	71.08±6.44	58.90±7.52
QS	98.80±0.38	89.30±0.76	76.65±0.51	95.17±0.45	82.38±3.12	85.88±1.13	86.02±0.97	65.47±8.37	*75.00±3.54*	69.08±2.87	*66.70±0.00*
Fisher_score	95.80±0.00	80.20±0.00	63.70±0.00	88.80±0.00	74.10±0.00	81.10±0.00	*89.50±0.00*	85.70±0.00	70.00±0.00	65.80±0.00	50.00±0.00
CIFE	71.20±0.00	82.90±0.00	61.60±0.00	59.60±0.00	44.60±0.00	71.80±0.00	68.40±0.00	57.10±0.00	70.00±0.00	71.10±0.00	44.40±0.00
ICAP	98.60±0.00	83.40±0.00	59.30±0.00	94.00±0.00	59.00±0.00	82.70±0.00	**91.70±0.00**	66.70±0.00	65.00±0.00	71.10±0.00	61.10±0.00
RFS	97.20±0.00	-	-	95.40±0.00	**87.20±0.00**	**90.30±0.00**	78.70±0.00	**90.50±0.00**	**85.00±0.00**	-	-
					ExtraTrees						
Baseline	100.0	96.9	87.39	96.51	94.04	93.59	96.99	85.71	82.5	78.95	69.44
NeuroFS	*99.94±0.12*	**93.68±0.43**	**84.26±0.55**	**95.44±0.27**	**91.46±0.73**	85.48±1.46	89.96±1.89	**90.50±0.00**	75.00±5.24	**81.75±4.16**	*75.46±6.71*
LassoNet	99.76±0.12	*92.96±0.15*	*83.68±0.13*	*94.86±0.22*	84.94±0.62	**91.12±0.30**	92.08±0.36	*89.54±1.92*	73.50±4.64	77.88±6.77	**76.12±3.80**
STG	**100.00±0.00**	90.38±0.42	82.05±0.48	94.32±0.21	88.50±2.15	88.68±0.42	83.56±1.62	83.84±3.80	*79.00±3.39*	75.78±5.10	71.08±2.24
QS	99.25±0.47	91.95±0.58	81.28±0.54	94.28±0.40	88.78±1.86	87.86±0.72	86.80±0.91	77.38±5.19	73.75±4.15	75.00±1.30	75.00±0.00
Fisher_score	96.86±0.43	84.86±0.15	72.06±0.08	90.94±0.24	81.42±0.59	85.50±0.30	*92.50±0.00*	**90.50±0.00**	60.00±1.58	74.22±1.95	63.90±0.00
CIFE	74.70±0.00	87.60±0.00	68.40±0.00	82.70±0.00	55.40±0.00	85.30±0.00	85.20±0.00	52.40±0.00	50.00±0.00	*81.60±0.00*	69.40±0.00
ICAP	99.70±0.00	87.80±0.00	65.50±0.00	93.50±0.00	70.60±0.00	89.20±0.00	**95.00±0.00**	81.00±0.00	**80.00±0.00**	78.90±0.00	63.90±0.00
RFS	98.30±0.00	-	-	94.70±0.00	*90.40±0.00*	*89.70±0.00*	86.50±0.00	**90.50±0.00**	75.00±0.00	-	-

most straightforward approach: at the end of the training process with RigL, we use neuron strength (same as in NeuroFS) on the trained network to derive the indices of the important features. We train a 3-layer MLP with Rigl for 100 epochs. RigL training algorithm updates the sparse connectivity at each epoch by removing ζ_h of the weights with the lowest magnitude and adding the same number of connections as the dropped ones to the network, among the non-existing connections with the highest gradient magnitude. When the training is finished we select the top K features corresponding to the neurons with the highest neuron strength as the selected features. The main difference between NeuroFS and feature selection using RigL is the input layer neuron removal and addition. While RigL updates only the sparse connectivity in all layers, NeuroFS updates also the neurons in the input layer to gradually decrease the number of

active neurons (neurons with at least one non-zero connection) to be suited for feature selection. All the experimental settings are similar to NeuroFS (Section 5.4.2), such as $\epsilon = 30$, $\zeta_h = 0.2$, training epochs, activation functions, batch size, learning rate, and etc. We measure the performance of feature selection using RigL for several values of $K \in \{25, 50, 75, 100, 150, 200\}$. The results are presented in Table D.3 and Figure D.2.

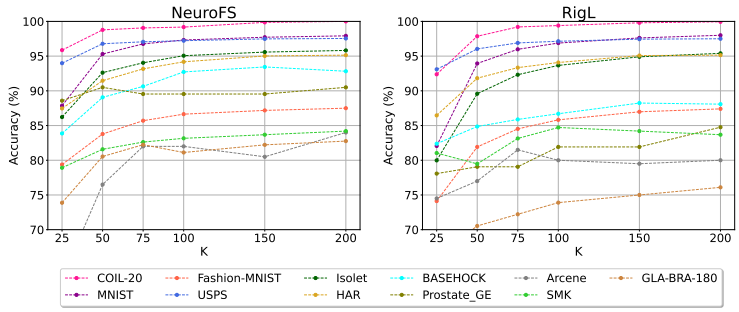


Figure D.2: NeuroFS (left) vs. feature selection using RigL (right) comparison.

As can be seen in Table D.3, NeuroFS outperforms feature selection with RigL in most cases in terms of classification accuracy. On low-dimensional datasets, RigL performs closely to NeuroFS and even outperforms it in some cases, particularly for large values of K; while for small values of K, e.g., 25 or 50, the performance gap is larger than the larger K values. On the other hand on high-dimensional datasets, NeuroFS outperforms RigL with a large gap in most datasets considered except SMK where they perform very closely. It can be concluded that the performance gap is usually high in cases where the proportion of selected features to the total number of features is low, e.g., selecting a low number of features in low-dimensional datasets and feature selection from high-dimensional datasets. Relatively similar behavior exists in feature selection using QuickSelection, particularly in high-dimensional datasets such as BASEHOCK, Prostate_GE, and GLA-BRA-180 (See Table D.4). The reason behind this is that when the search space becomes large (all input features), finding a low fraction of informative features becomes difficult for QuickSelection and RigL and the neuron strength might not be informative on its own. Therefore, NeuroFS reduces the search space by removing uninformative features during training, thus allowing the high-magnitude weights to be assigned to a limited set of the most informative features. This indicates the importance of the neuron removal

scheme in NeuroFS.

Table D.3: Supervised feature selection comparison (classification accuracy (%)) with RigL for $K \in \{25, 50, 75, 100, 150, 200\}$. Bold fonts indicate the best performer for each dataset.

Dataset	Method	$K = 25$	$K = 50$	$K = 75$	$K = 100$	$K = 150$	$K = 200$
COIL-20	NeuroFS	**95.86 ± 1.31**	**98.78 ± 0.29**	99.06 ± 0.12	99.18 ± 0.5	**99.86 ± 0.28**	**100.0 ± 0.0**
	RigL	92.38 ± 3.2	97.86 ± 1.32	**99.2 ± 0.43**	**99.4 ± 0.43**	99.8 ± 0.28	99.94 ± 0.12
MNIST	NeuroFS	**87.86 ± 1.77**	**95.3 ± 0.41**	**96.76 ± 0.22**	**97.32 ± 0.17**	**97.72 ± 0.1**	97.92 ± 0.07
	RigL	82.06 ± 0.99	93.94 ± 0.63	95.98 ± 0.51	96.88 ± 0.22	97.62 ± 0.07	**98.0 ± 0.09**
Fashion-MNIST	NeuroFS	**79.38 ± 0.96**	**83.78 ± 0.64**	**85.7 ± 0.28**	**86.64 ± 0.21**	**87.18 ± 0.16**	**87.5 ± 0.17**
	RigL	74.12 ± 1.59	81.92 ± 0.87	84.52 ± 0.72	85.82 ± 0.23	86.98 ± 0.12	87.4 ± 0.23
USPS	NeuroFS	**93.98 ± 0.87**	**96.78 ± 0.17**	**97.06 ± 0.15**	**97.22 ± 0.12**	**97.48 ± 0.04**	**97.54 ± 0.1**
	RigL	93.1 ± 0.62	96.04 ± 0.58	96.9 ± 0.24	97.14 ± 0.1	97.44 ± 0.1	97.5 ± 0.11
Isolet	NeuroFS	**86.22 ± 0.84**	**92.62 ± 0.4**	**94.04 ± 0.34**	**95.06 ± 0.31**	**95.58 ± 0.29**	**95.82 ± 0.31**
	RigL	79.98 ± 2.25	89.58 ± 1.24	92.32 ± 0.56	93.66 ± 0.58	94.9 ± 0.56	95.4 ± 0.32
HAR	NeuroFS	**87.46 ± 0.79**	91.46 ± 0.72	93.16 ± 0.79	**94.18 ± 0.29**	95.02 ± 0.35	**95.14 ± 0.21**
	RigL	86.46 ± 1.47	**91.82 ± 0.3**	**93.34 ± 0.47**	94.08 ± 0.26	**95.08 ± 0.26**	**95.14 ± 0.37**
BASEHOCK	NeuroFS	**83.86 ± 3.38**	**89.06 ± 2.46**	**90.64 ± 2.35**	**92.72 ± 1.5**	**93.44 ± 0.91**	**92.82 ± 1.13**
	RigL	82.38 ± 2.85	84.86 ± 3.04	85.88 ± 2.32	86.7 ± 1.54	88.24 ± 1.7	88.08 ± 1.92
Prostate_GE	NeuroFS	**88.58 ± 2.35**	**90.5 ± 0.0**	**89.54 ± 1.92**	**89.54 ± 1.92**	**89.54 ± 1.92**	**90.5 ± 0.0**
	RigL	78.08 ± 6.46	79.06 ± 7.11	79.06 ± 8.83	81.92 ± 8.18	81.92 ± 6.33	84.76 ± 1.88
Arcene	NeuroFS	63.0 ± 4.85	76.5 ± 2.55	**82.0 ± 4.0**	**82.0 ± 1.87**	**80.5 ± 4.3**	**84.0 ± 3.39**
	RigL	**74.5 ± 4.3**	**77.0 ± 3.32**	81.5 ± 4.64	80.0 ± 4.47	79.5 ± 4.3	80.0 ± 4.18
SMK	NeuroFS	78.92 ± 1.68	**81.58 ± 1.68**	82.62 ± 2.12	83.16 ± 1.27	83.68 ± 1.04	**84.2 ± 0.0**
	RigL	**81.04 ± 1.99**	79.48 ± 4.81	**83.14 ± 3.15**	**84.72 ± 1.95**	**84.2 ± 3.73**	83.68 ± 2.55
GLA-BRA-180	NeuroFS	**73.88 ± 3.8**	**80.54 ± 4.96**	**82.24 ± 3.31**	**81.12 ± 2.05**	**82.22 ± 1.32**	**82.76 ± 2.71**
	RigL	66.1 ± 3.22	70.54 ± 4.16	72.22 ± 4.98	73.9 ± 3.76	75.0 ± 3.07	76.1 ± 4.16

D.5 Comparison Results

The detailed results for each K value are presented in Table D.4.

Table D.4: Supervised feature selection comparison (classification accuracy for various K values (%)). Empty entries show that the corresponding experiments exceeded the time limit (12 hours). Bold and italic fonts indicate the best and second-best performer, respectively.

(a) K = 25

	Low-dimensional Datasets						High-dimensional Datasets				
Method	**COIL-20**	**MNIST**	**Fashion-MNIST**	**USPS**	**Isolet**	**HAR**	**BASEHOCK**	**Prostate_GE**	**Arcene**	**SMK**	**GLA-BRA-180**
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	*95.86±1.31*	**87.86±1.77**	**79.38±0.96**	93.98±0.87	**86.22±0.84**	87.46±0.79	83.86±3.38	*88.58±2.35*	63.00±4.85	*78.92±1.68*	*73.88±3.80*
LassoNet	92.72±0.85	*86.40±1.26*	*78.68±0.55*	*94.04±0.38*	76.48±0.39	**93.00±0.31**	84.48±0.86	*88.58±2.35*	69.00±2.55	76.84±5.34	**76.12±4.19**
STG	**97.02±1.41**	85.24±1.89	77.44±0.53	*94.04±0.46*	*77.16±4.34*	87.48±0.80	82.38±1.36	85.72±3.00	69.00±5.15	77.38±3.57	67.22±4.78
QS	91.00±4.21	85.25±1.47	71.57±1.97	93.00±0.81	72.56±6.53	87.14±1.74	83.80±1.61	71.43±12.16	*73.75±8.20*	76.97±7.52	69.45±2.75
Fisher_score	24.70±0.00	74.40±0.00	53.10±0.00	82.00±0.00	57.40±0.00	77.10±0.00	*85.50±0.00*	**90.50±0.00**	65.00±0.00	68.40±0.00	58.30±0.00
CIFE	50.70±0.00	80.90±0.00	63.40±0.00	50.20±0.00	56.00±0.00	80.20±0.00	76.20±0.00	61.90±0.00	67.50±0.00	**81.60±0.00**	61.10±0.00
ICAP	94.40±0.00	81.60±0.00	50.10±0.00	89.90±0.00	67.10±0.00	84.50±0.00	**89.20±0.00**	47.60±0.00	**77.50±0.00**	78.90±0.00	69.40±0.00
RFS	88.20±0.00	-	-	**94.80±0.00**	76.50±0.00	*88.90±0.00*	80.20±0.00	**90.50±0.00**	**77.50±0.00**	-	-

(b) K = 50

	Low-dimensional Datasets						High-dimensional Datasets				
Method	**COIL-20**	**MNIST**	**Fashion-MNIST**	**USPS**	**Isolet**	**HAR**	**BASEHOCK**	**Prostate_GE**	**Arcene**	**SMK**	**GLA-BRA-180**
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	98.78±0.29	**95.30±0.41**	**83.78±0.64**	**96.78±0.17**	**92.62±0.40**	91.46±0.72	*89.06±2.46*	**90.50±0.00**	*76.50±2.55*	*81.58±1.68*	**80.54±4.96**
LassoNet	97.16±1.06	*94.46±0.21*	*82.58±0.10*	95.94±0.15	84.90±0.22	*93.74±0.39*	87.18±0.58	*88.58±2.35*	71.00±2.00	80.52±2.69	*74.46±4.78*
STG	**99.32±0.40**	93.20±0.62	82.36±0.52	*96.62±0.34*	85.82±2.83	91.22±1.23	85.12±1.86	84.78±3.55	71.00±2.55	80.25±2.95	70.00±4.08
QS	96.52±1.53	93.62±0.49	80.82±0.51	95.52±0.27	89.78±1.80	91.96±1.04	87.22±1.22	76.20±7.53	74.38±4.80	80.90±2.20	72.20±2.80
Fisher_score	74.00±0.00	81.90±0.00	67.80±0.00	91.00±0.00	67.40±0.00	79.80±0.00	**90.20±0.00**	**90.50±0.00**	67.50±0.00	73.70±0.00	63.90±0.00
CIFE	59.40±0.00	89.30±0.00	66.90±0.00	61.30±0.00	59.80±0.00	84.20±0.00	77.40±0.00	47.60±0.00	52.50±0.00	**81.60±0.00**	58.30±0.00
ICAP	*99.30±0.00*	89.00±0.00	59.50±0.00	95.20±0.00	75.10±0.00	88.70±0.00	**90.20±0.00**	57.10±0.00	70.00±0.00	73.70±0.00	72.20±0.00
RFS	95.80±0.00	-	-	95.80±0.00	*91.50±0.00*	**94.00±0.00**	85.00±0.00	**90.50±0.00**	**77.50±0.00**	-	-

(c) K = 75

	Low-dimensional Datasets						High-dimensional Datasets				
Method	**COIL-20**	**MNIST**	**Fashion-MNIST**	**USPS**	**Isolet**	**HAR**	**BASEHOCK**	**Prostate_GE**	**Arcene**	**SMK**	**GLA-BRA-180**
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	99.06±0.12	**96.76±0.22**	**85.70±0.28**	*97.06±0.15*	**94.04±0.34**	93.16±0.79	*90.64±2.35*	*89.54±1.92*	**82.00±4.00**	**82.62±2.12**	**82.24±3.31**
LassoNet	99.46±0.35	*96.00±0.09*	83.92±0.13	96.36±0.08	91.00±0.62	*94.62±0.17*	90.52±0.27	**90.50±0.00**	70.50±2.45	78.94±3.72	*76.64±5.44*
STG	*99.68±0.22*	95.52±0.22	*84.14±0.43*	96.88±0.23	90.10±2.17	92.42±1.11	85.52±1.22	84.78±3.55	75.00±2.74	81.04±4.21	71.08±1.37
QS	98.17±1.16	95.98±0.33	83.80±0.53	96.85±0.05	93.04±0.46	93.50±0.77	87.55±1.30	72.62±9.78	76.88±2.72	*82.22±2.86*	73.60±1.40
Fisher_score	76.00±0.00	87.10±0.00	74.30±0.00	94.40±0.00	76.00±0.00	81.70±0.00	89.00±0.00	**90.50±0.00**	70.00±0.00	76.30±0.00	66.70±0.00
CIFE	63.20±0.00	92.70±0.00	67.70±0.00	68.00±0.00	74.30±0.00	84.80±0.00	74.70±0.00	47.60±0.00	72.50±0.00	76.30±0.00	58.30±0.00
ICAP	99.00±0.00	92.40±0.00	67.20±0.00	95.30±0.00	79.70±0.00	89.20±0.00	**93.50±0.00**	57.10±0.00	72.50±0.00	71.10±0.00	72.20±0.00
RFS	**99.70±0.00**	-	-	**97.20±0.00**	*93.90±0.00*	**94.90±0.00**	86.50±0.00	**90.50±0.00**	*80.00±0.00*	-	-

(d) K = 100

Method	Low-dimensional Datasets						High-dimensional Datasets				
	COIL-20	MNIST	Fashion-MNIST	USPS	Isolet	HAR	BASEHOCK	Prostate_GE	Arcene	SMK	GLA-BRA-180
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	99.18±0.50	**97.32±0.17**	**86.64±0.21**	*97.22±0.12*	**95.06±0.31**	94.18±0.29	*92.72±1.50*	*89.54±1.92*	*82.00±1.87*	*83.16±1.27*	**81.12±2.05**
LassoNet	99.30±0.00	96.64±0.14	84.98±0.18	97.04±0.12	93.18±0.22	*95.14±0.29*	90.96±1.36	**90.50±0.00**	72.00±4.30	78.42±4.20	*79.46±2.83*
STG	*99.76±0.12*	96.38±0.35	85.20±0.58	97.08±0.18	92.64±0.56	92.82±0.74	85.96±1.24	85.72±3.00	75.50±3.67	82.08±3.87	72.20±3.07
QS	98.28±1.15	*96.85±0.09*	*85.52±0.15*	97.00±0.14	94.22±0.28	94.06±0.48	89.02±1.26	78.58±9.82	78.12±1.08	**84.85±2.16**	73.60±1.40
Fisher_score	80.20±0.00	90.70±0.00	79.60±0.00	96.50±0.00	79.80±0.00	83.80±0.00	89.70±0.00	**90.50±0.00**	65.00±0.00	78.90±0.00	66.70±0.00
CIFE	67.70±0.00	95.10±0.00	69.20±0.00	78.00±0.00	81.20±0.00	85.30±0.00	74.40±0.00	71.40±0.00	65.00±0.00	81.60±0.00	58.30±0.00
ICAP	**100.00±0.00**	95.00±0.00	77.70±0.00	95.40±0.00	82.80±0.00	92.10±0.00	**94.00±0.00**	52.40±0.00	**82.50±0.00**	76.30±0.00	69.40±0.00
RFS	**100.00±0.00**	-	-	**97.40±0.00**	*94.40±0.00*	**95.40±0.00**	86.70±0.00	**90.50±0.00**	80.00±0.00	-	-

(e) K = 150

Method	Low-dimensional Datasets						High-dimensional Datasets				
	COIL-20	MNIST	Fashion-MNIST	USPS	Isolet	HAR	BASEHOCK	Prostate_GE	Arcene	SMK	GLA-BRA-180
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	99.86±0.28	**97.72±0.10**	**87.18±0.16**	**97.48±0.04**	*95.58±0.29*	95.02±0.35	**93.44±0.91**	*89.54±1.92*	**80.50±4.30**	*83.68±1.04*	**82.22±1.32**
LassoNet	99.54±0.20	97.42±0.07	86.02±0.15	**97.48±0.07**	94.96±0.15	**95.72±0.20**	92.58±0.62	**90.50±0.00**	72.00±1.87	78.38±1.04	*79.48±1.37*
STG	**100.00±0.00**	97.14±0.08	86.28±0.35	97.28±0.12	94.20±0.35	93.56±0.59	86.42±1.74	84.76±4.67	75.00±3.16	82.60±4.27	72.76±3.27
QS	*99.92±0.13*	*97.70±0.12*	*86.88±0.32*	*97.42±0.11*	95.48±0.32	94.80±0.24	89.80±0.83	79.75±6.19	78.75±1.25	**84.22±3.23**	75.00±0.00
Fisher_score	81.20±0.00	93.10±0.00	83.60±0.00	97.30±0.00	83.00±0.00	84.40±0.00	91.70±0.00	**90.50±0.00**	65.00±0.00	78.90±0.00	63.90±0.00
CIFE	71.90±0.00	96.80±0.00	75.60±0.00	89.60±0.00	85.70±0.00	85.90±0.00	79.20±0.00	76.20±0.00	55.00±0.00	81.60±0.00	72.20±0.00
ICAP	**100.00±0.00**	96.40±0.00	81.70±0.00	95.80±0.00	89.30±0.00	93.40±0.00	*93.20±0.00*	42.90±0.00	77.50±0.00	71.10±0.00	69.40±0.00
RFS	**100.00±0.00**	-	-	97.40±0.00	**95.90±0.00**	*95.50±0.00*	88.20±0.00	**90.50±0.00**	*80.00±0.00*	-	-

(f) K = 200

Method	Low-dimensional Datasets						High-dimensional Datasets				
	COIL-20	MNIST	Fashion-MNIST	USPS	Isolet	HAR	BASEHOCK	Prostate_GE	Arcene	SMK	GLA-BRA-180
Baseline	100.0	97.92	88.3	97.58	96.03	95.05	91.98	80.95	77.5	86.84	72.22
NeuroFS	**100.00±0.00**	*97.92±0.07*	*87.50±0.17*	*97.54±0.10*	*95.82±0.31*	95.14±0.21	92.82±1.13	**90.50±0.00**	**84.00±3.39**	**84.20±0.00**	**82.76±2.71**
LassoNet	**100.00±0.00**	97.90±0.00	86.66±0.14	**97.58±0.07**	95.34±0.05	*95.58±0.12*	*92.92±1.06*	**90.50±0.00**	73.50±2.55	78.92±1.68	*80.02±2.06*
STG	**100.00±0.00**	97.46±0.20	87.00±0.26	97.38±0.04	94.88±0.07	93.70±0.66	87.46±1.03	*86.68±3.56*	77.00±3.32	81.54±3.34	73.88±3.80
QS	*99.50±0.53*	**98.00±0.07**	**87.52±0.15**	97.50±0.00	**96.14±0.08**	94.76±0.29	90.18±0.66	79.75±6.19	*80.62±3.25*	*82.88±5.44*	73.60±1.40
Fisher_score	84.00±0.00	94.50±0.00	84.70±0.00	97.50±0.00	89.90±0.00	85.80±0.00	92.20±0.00	**90.50±0.00**	65.00±0.00	78.90±0.00	61.10±0.00
CIFE	72.20±0.00	97.60±0.00	78.80±0.00	96.30±0.00	87.90±0.00	85.90±0.00	79.20±0.00	76.20±0.00	57.50±0.00	78.90±0.00	72.20±0.00
ICAP	99.30±0.00	97.60±0.00	84.50±0.00	96.90±0.00	90.30±0.00	93.30±0.00	**93.70±0.00**	42.90±0.00	80.00±0.00	76.30±0.00	72.20±0.00
RFS	**100.00±0.00**	-	-	97.50±0.00	95.70±0.00	**95.80±0.00**	89.00±0.00	**90.50±0.00**	80.00±0.00	-	-

Curriculum Vitae

Zahra Atashgahi was born on January 7, 1995, in Neyshabour, Iran. She completed her B.Sc. in Computer Engineering at Amirkabir University of Technology (Tehran Polytechnic) in 2017. Following this, she was awarded direct admission to the graduate program at the same university, where she pursued M.Sc. in Artificial Intelligence from 2017 to 2019. Her M.Sc. thesis focused on abnormal activity detection from the daily life of people suffering from dementia.

In October 2019, Zahra moved to the Netherlands, where she started her Ph.D. in the Data Mining group, Mathematics and Computer Science Department, at the Eindhoven University of Technology. In May 2020, she moved to the University of Twente, to continue her Ph.D. in the Data Management and Biometric group, at the Faculty of Electrical Engineering, Mathematics and Computer Science. Throughout her doctoral studies, she was under the expert guidance of Dr. Decebal Mocanu, Prof. Dr. Raymond Veldhuis, and Prof. Dr. Mykola Pechenizkiy. Zahra also actively contributed to the EDIC (Exceptional and Deep Intelligent Coach) project, which aims to develop an intelligent coach for promoting a healthy lifestyle.

Throughout her academic journey, Zahra actively engaged in the scientific community. She served as a program committee member for prestigious conferences such as NeurIPS, ICML, and AAAI. She served twice as a student volunteer at IJCAI. She has organized a study group on mathematics and machine learning at the University of Twente. She co-organized a workshop on sparse neural networks at ICLR, tutorials on SNNs at ECMLPKDD and IJCAI, and an online study group on SNNs. She did a research visit at the van der Schaar Lab, Department of Applied Mathematics and Theoretical Physics at the University of Cambridge under the supervision of Prof. Dr. Mihaela van der Schaar. Towards the end of her Ph.D., she did an internship at Booking.com as a machine learning scientist intern working on data-driven decision-making for marketing tasks.

List of Publications

The list of all publications and preprints resulting from this PhD research is included below.

Journal Publications [Peer-reviewed]

1. **Zahra Atashgahi**, Xuhao Zhang, Neil Kichler, Shiwei Liu, Lu Yin, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu. *Supervised feature selection with neuron evolution in sparse neural networks*. Transactions on Machine Learning Research (TMLR), 2023. (This paper contributes to Chapter 5 of the thesis.)

2. **Zahra Atashgahi**, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. *Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders*. Machine Learning 111, ECML-PKDD journal track, 377–414, 2022. (This paper contributes to Chapter 4 of the thesis.)

3. **Zahra Atashgahi**, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. *A brain-inspired algorithm for training highly sparse neural networks.* Machine Learning 111, ECML-PKDD journal track, 4411-4452, 2022. (This paper contributes to Chapter 2 of the thesis.)

Conference Publications [Peer-reviewed]

4. Kaiting Liu, **Zahra Atashgahi**, Ghada Sokar, Mykola Pechenizkiy, Decebal Constantin Mocanu. *Supervised Feature Selection via Ensemble Gradient In-*

formation from Sparse Neural Networks. Artificial Intelligence and Statistics Conference (AISTATS), 2024. (This paper is a follow-up of Chapter 4 of the thesis.)

5. **Zahra Atashgahi**, *Cost-effective Artificial Neural Networks*, International Joint Conferences on Artificial Intelligence Organization (IJCAI), Doctoral Consortium, 2023. (This paper contributes to Chapter 1 of the thesis.)

6. Ghada Sokar, **Zahra Atashgahi**, Mykola Pechenizkiy, and Decebal Constantin Mocanu. *Where to pay attention in sparse training for feature selection?*. Advances in Neural Information Processing Systems (NeurIPS), 2023. (This paper is a follow-up of Chapter 4 of the thesis.)

7. Shiwei Liu, Tianlong Chen, **Zahra Atashgahi**, Xiaohan Chen, Ghada Sokar, Elena Mocanu, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. *Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity*. International Conference on Learning Representations (ICLR), 2022.

8. Shiwei Liu, Tianlong Chen, Xiaohan Chen, **Zahra Atashgahi**, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. *Sparse Training via Boosting Pruning Plasticity with Neuroregeneration*. Advances in Neural Information Processing Systems (NeurIPS), 2021.

9. Shiwei Liu, Tim Van der Lee, Anil Yaman, **Zahra Atashgahi**, Davide Ferraro, Ghada Sokar, Mykola Pechenizkiy, and Decebal Constantin Mocanu. *Topological insights into sparse neural networks.* The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, (ECML-PKDD), Ghent, Belgium, 2020.

Preprints [under review]

10. **Zahra Atashgahi**, Tennison Liu, Mykola Pechenizkiy, Raymond Veldhuis, Decebal Constantin Mocanu, Mihaela van der Schaar. *Unveiling the Power*

of Sparse Neural Networks for Feature Selection. (under review at ECAI 2024), 2024. (This paper is a follow-up of Chapters 4 and 5 of the thesis.)

11. **Zahra Atashgahi**, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu. *Adaptive Sparsity Level during Training for Efficient Time Series Forecasting with Transformers*. arXiv preprint arXiv:2305.18382 (under review at ECML-PKDD 2024), 2023. (This paper contributes to Chapter 3 of the thesis.)

12. **Zahra Atashgahi**, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. *Memory-free online change-point detection: A novel neural network approach*. arXiv preprint arXiv:2207.03932 (Under review at Neural Computing and Applications journal, current status: major revision), 2022.

Wokshop Papers & Posters [Peer-reviewed]

13. **Zahra Atashgahi**, Xuhao Zhang, Neil Kichler, Shiwei Liu, Lu Yin, Mykola Pechenizkiy, Raymond Veldhuis, and Decebal Constantin Mocanu. *Feature selection with neuron evolution in sparse neural networks*. ICLR 2023 Workshop on Sparsity in Neural Networks: On practical limitations and tradeoffs between sustainability and efficiency, Kigali, Rwanda, 2023.

14. **Zahra Atashgahi**, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. *A brain-inspired algorithm for training highly sparse neural networks*. Sparsity in Neural Networks: Advancing Understanding and Practice, 2022.

15. **Zahra Atashgahi**, *Cost-effective artificial neural networks*. International Joint Conferences on Artificial Intelligence Organization (IJCAI), doctoral consortium, 2021.

16. **Zahra Atashgahi**, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. *Quick*

and robust feature selection: The strength of energy-efficient sparse training for autoencoders. Sparsity in Neural Networks: Advancing Understanding and Practice, 2021.

17. Neil Kichler, **Zahra Atashgahi**, Decebal Constantin Mocanu. *Robustness of sparse MLPs for supervised feature selection*. Sparsity in Neural Networks: Advancing Understanding and Practice, 2021.

18. Shiwei Liu, Tianlong Chen, **Zahra Atashgahi**, Xiaohan Chen, Ghada Sokar, Elena Mocanu, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. *Freetickets: Accurate, robust and efficient deep ensemble by training with dynamic sparsity*. Sparsity in Neural Networks: Advancing Understanding and Practice, 2021.

19. **Zahra Atashgahi**, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Ramond Veldhuis, and Mykola Pechenizkiy. *Quick and Robust Feature Selection: the Strength of Energy-efficient Sparse Training for Autoencoders (Extended Abstract)*. Joint International Scientific Conferences on AI BNAIC/BENELEARN, 2021.

20. **Zahra Atashgahi**, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. *Unsupervised online memory-free change-point detection using an ensemble of LSTM-autoencoder-based neural networks*. In 8th ACM Celebration of Women in Computing womENcourage, 2021.

Acknowledgments

The journey to completing this thesis has been one of immense growth, learning, and discovery, made possible by the support, guidance, and encouragement of numerous individuals to whom I owe my sincere gratitude.

First and foremost, I am immensely grateful to my supervisors, Prof. dr. R. Veldhuis, Prof. dr. Mykola Pechenizkiy, and Prof. dr. Decebal Mocanu, for their unwavering support, invaluable guidance, and insightful feedback throughout my research journey. Their mentorship has not only shaped my academic development but also prepared me for the challenges of professional life. I am particularly thankful for the freedom I was given to explore my research interests, which has been crucial in shaping my path forward. I want to express my heartfelt gratitude to Decebal for his belief in my abilities, continuous guidance, and invaluable advice throughout my Ph.D. journey. His continuous support, encouragement, and motivation have been instrumental in shaping both my research endeavors and professional growth. Additionally, I am immensely grateful for the opportunities Decebal created for me to co-organize tutorials, workshops, and supervise students, experiences that have enriched my journey and expanded my network. Thank you, Decebal, for acknowledging every achievement and effort, and offering unwavering assistance and support. Furthermore, I extend my appreciation to Raymond for our regular discussions, his invaluable advice, and his support in my research pursuits. Our conversations, filled with enthusiasm and critical analysis, have opened new views of knowledge and inquiry for me. Your willingness to delve deep into the nuances of my work has been invaluable in refining my ideas and methodologies. Moreover, I am particularly thankful to Mykola for his insightful guidance, the experiences we've shared at conferences, and the unforgettable hikes during these events. Mykola has taught me critical thinking, offered constructive feedback, and facilitated my professional growth and networking. I am deeply appreciative of having such amazing supervisors.

I would like to express my sincere gratitude to the committee members, Prof. Dr. Christin Seifert from the University of Marburg, Dr. Matthew Taylor from the University of Alberta, Dr. Cassio de Campos from the Eindhoven University of Technology, Prof. Dr. Christoph Brune from the University of Twente, and Prof. Dr. Johannes Schmidt-Hieber from the University of Twente, for their invaluable contributions to this Ph.D. defense. I am truly thankful for the time and effort you have dedicated to reviewing my thesis.

To my paranymphs Isil and Boqian, I want to thank you for taking a special place in my ceremony. Isil, your warm and friendly personality and ever-present smile have been a delight. It has been a pleasure to have you as both a colleague and friend. I cherish our discussions in the office and the enjoyable coffee breaks we shared. Boqian, your kindness and friendliness have been a source of support. I am thankful for the delightful experiences we've had at conferences together. Your willingness to help whenever I needed it has been greatly comforting.

I am thankful for the opportunity to be a part of the DMB research group at the University of Twente. The group events, lunch talks, and the friendly, supportive atmosphere have been invaluable. I extend my gratitude to the group secretaries, Bertine, Karen, and Laura, whose patience, assistance, and support at various stages of my PhD journey have been instrumental. Their efficient work and care have made me feel at home even while studying in a new environment. I also want to express my appreciation to Isil, Tugce, Jeroen, and Faryal, with whom I shared an office. Our discussions ranged from academic matters to the joys and challenges of pursuing a Ph.D., making our time together enriching. Special thanks to Isil, Tugce, Shunxin, and Shreyasi for our engaging discussions in the mathematics study group and the memorable trip we took to Dusseldorf, the moments we shared, and our conversations during coffee breaks, where we could openly share our experiences, joys, and frustrations of Ph.D. life. It has been an absolute pleasure to have you as colleagues and friends.

I am deeply grateful for the opportunity to be part of the data mining group at Eindhoven University of Technology during the initial months of my PhD. I extend my heartfelt thanks to our group's secretary, José. Her patience, assistance with complex paperwork, and support when I moved to the Netherlands were invaluable. I also want to thank Mykola for arranging the Dutch game night and the ICLR local event at Eindhoven University of Technology and inviting me to present there. Additionally, I'm thankful to all my colleagues for the friendly atmosphere, engaging coffee breaks, and fun lunch talks.

I'm grateful for being part of the BlueNN group meetings organized by Decebal, alongside group members Tim, Anil, Shiwei, Lu, Ghada, Bram, Qiao, Boqian, Aleksandra, Christopher, Ines, and Farid. Our weekly meetings, whether

in person or virtual, and group events were cherished moments throughout my PhD journey. Sharing life and academic experiences during these gatherings was heartwarming, and the insightful discussions we had were invaluable and inspiring. Special thanks to my co-authors Elena, Shiwei, Ghada, and Lu for our collaborative work; the learning experience was truly precious. I'm thankful to have you all as friends and colleagues. In particular, I extend my heartfelt thanks to Decebal and Elena for their efforts in organizing various team activities, conference trips, and social gatherings, including their warm hospitality at their home, where they made us feel welcome.

I am deeply grateful for the opportunity to be a part of the EDIC project under the supervision of Prof. Dr. Hermie Hermens. Collaborating with the researchers within the EDIC consortium has allowed me to broaden my perspective, learn from experts, apply my knowledge to the medical field, and gain invaluable insights from medical professionals. I extend my heartfelt thanks to the dynamic PhD group in EDIC, including Carlijn, Rianne, Niala, Ellen, and Vishnu, for our discussions, presentations, and the organization of team outings and meetings.

I am grateful for the invaluable opportunity to do a research visit to the group led by Prof. Dr. Mihaela van der Schaar at the University of Cambridge. I extend my heartfelt thanks to Mihaela for not only providing me with this enriching experience but also for her exceptional guidance, which has significantly contributed to my professional growth. Our discussions were consistently enlightening, offering me profound insights into the intricacies of our field. I am particularly grateful for the memorable invitation to London and our delightful excursion to a museum. I deeply admire Mihaela's dedication to supporting women in science. Additionally, I express my gratitude to Tennison for our collaborative efforts and the stimulating discussions that broadened my understanding. I would also like to extend my appreciation to all the members of the van der Schaar Lab for their warmth and support throughout my time in Cambridge.

I express my sincere gratitude for the invaluable internship experience I had at Customer Marketing Machine Learning at booking.com, under the guidance of Eva (Anna) Schmit and Yuan Lu. Thank you, Anna and Yuan, for your daily guidance and discussions were indispensable in navigating through the project, and I am deeply thankful for your continuous support throughout this incredible journey. Special acknowledgment goes to Aytekin, our group manager, for his consistent guidance, sharing his insightful business perspectives, and support during my internship. I am also grateful to Niklas and Dima for their help in the project and technical assistance and to the entire team for fostering a friendly and supportive environment filled with engaging discussions, delightful lunch talks, and enjoyable coffee breaks. The organization of the outing dinner and the

farewell gathering added an unforgettable touch to this experience. Additionally, I extend my appreciation to the Ph.D. internship organizing team for arranging enriching social events with fellow interns and mentors, adding further value to my overall internship journey.

I extend my deepest gratitude to my beloved friends for their unwavering presence throughout this journey. Maryam, your friendship has been a source of joy and strength, and I am thankful for your cheerful personality, our heartfelt conversations, and your constant support, regardless of distance. Noushin and Hamed, I am grateful for the incredible moments we've shared, from our adventurous trips and hikes to our joyful meals, fun outings, endless talking, and laughter. Sandra and Matias, thank you for your continuous encouragement, listening, advising, and offering help and support. Your companionship has brought countless memorable experiences, from our engaging conversations to our shared meals, board game nights, outings, exercises, cycling, and ceramics workshops. To my cousin and friend, Sara, thank you for being there for me since our childhood no matter of distance, and for the wonderful time we spent together. Azadeh and Naser, thank you for all the fun activities we had together and the memorable moments we built.

My journey would not have been the same without the love, patience, and understanding of my family. To my parents, whose unconditional love and sacrifices have paved the way for my achievements, I owe an immeasurable debt of gratitude. To my mother, your unwavering belief in my dreams has been a guiding light, providing me with strength and inspiration at every turn. Your own pursuit of dreams serves as a continual source of motivation. To my father, I extend heartfelt thanks for your tireless efforts to ease my path. To my dearest sister, Reyhaneh, thank you for making my days brighter and more joyful. To my beloved partner, Ahmad, your endless support, encouragement, and love have been my anchor through every challenge. Your belief in my dreams has fueled my determination to persevere through the toughest of times. Thank you for being beside me through every high and low. Your belief in me, even when I doubted myself, fueled my determination, transforming this achievement into a shared triumph. Thank you for always listening to me, offering your advice, and for your cheerful nature and positivity, which have consistently brightened my days. I am immensely grateful to have had you by my side throughout this journey. Here's to the future and the adventures that await us.

Zahra Atashgahi
Amersfoort, April, 2024