Survival of the Most Connected: An Understanding of How Sparse Neural Networks Grow

Dylan Matab [*]
University of Twente
d.s.k.d.matab@student.utwente.nl

Elena Mocanu[†]
University of Twente
e.mocanu@utwente.nl

Boqian Wu[†]
University of Twente
b.wu@utwente.nl

Abstract

Artificial neural networks excel at complex tasks; however, scaling them increases parameters and computational power disproportionately, often without corresponding performance gains. Sparse neural networks, which have significantly fewer connections, address this issue. Dynamic Sparse Training facilitates the evolution of sparse networks through an iterative process of pruning and regrowing connections. While pruning strategies are well-studied, regrow strategies are less understood. We present an analysis of state-of-the-art regrow strategies, examining their effects on performance and network structure. Our study reveals that gradient-based regrow methods quickly create highly connected neurons, random regrow methods promote uniform connectivity, and momentum-based methods offer a balance between speed and stability. These findings highlight the trade-offs in speed and stability among different regrow strategies. By integrating performance metrics and network properties from initialization to post-training, our results provide valuable insights for designing efficient sparse neural network training procedures. This research enhances the sparse-to-sparse training paradigm and contributes to the understanding of sparse network dynamics, including connectivity patterns and neuron activation. Our findings benefit the fields of neural network optimization, network science, and machine learning.

1 Introduction

An essential part of Dynamic Sparse Training (DST) is is the process of *pruning* and *regrowing* connections between neurons. The current literature provides extensive analysis on pruning strategies [42, 27, 31, 33], but not on regrow strategies. Iteratively changing the topology of a network can be seen as a mutation process, whereby a sparse neural network evolves over time during training. In simpler terms, the restructuring and reorganization of the network improve its ability to learn the given task. In this evolutionary process, the chosen regrow strategy plays a vital role in how efficiently and effectively the network learns.

When pruning connections pre-training, there is a possibility that important nodes lose connections at the initialization phase [14]. This implies that, along with pruning, regrowing connections also plays an important role in the training process. As far as we know, there is no extensive comparative analysis on the effects of various **regrowing** strategies, specifically studies that include both performance and a deep understanding of topological formation. Studies have highlighted the critical importance

[*]Corresponding author of this report
[†]Research carried out under the supervision of

of pruning in the evolution of network topology and performance [42, 27]. However, the lack of similar detailed studies on regrow strategies presents a gap in the current understanding of their effects. Addressing this gap is crucial because different regrow strategies can significantly impact the training efficiency and effectiveness of sparse neural networks. Our motivation is to provide a comprehensive comparative analysis of regrow strategies, thereby enhancing the sparse-to-sparse training procedure and contributing to the broader understanding of sparse neural network dynamics, including connectivity patterns, neuron activation, and network robustness.

Our goal is to provide a better understanding of how various **regrow** strategies affect performance and how they influence the evolution process and structure of an artificial neural network. This empirical study spans across different datasets in mostly high sparsity regimes. High sparsity regimes are chosen because they represent the most challenging and informative scenarios for understanding the behavior of sparse networks, particularly regarding how network connectivity and learning efficiency are maintained under extreme sparsity conditions [20, 16]. Considering our results, in this study, we present multiple contributions. Our contributions span fields such as neural network optimization, network science, and machine learning, all of which relate back to the behavior and efficiency of sparse neural networks.

(A) Topological Connectivity and Preferential Attachment Firstly, we analyze the journey sparse neural networks take when exploring topological landscapes, and how state-of-the-art regrow strategies affect the exploration process. This analysis measures how the networks develop over time to learn the given task. Based on our empirical results, particularly the analysis of the distribution of connections between neurons, we show the occurrence of *preferential attachment*, thereby connecting the Barabasi-Albert model [6] to sparse neural networks, showing that their characteristics align, regardless of the choice of regrow strategy. We formulate:

The Neural Selection Hypothesis. *The hidden layers of sparse neural networks exhibit the preferential attachment mechanism, whereby, during sparse-to-sparse training within the DST framework, using unstructured pruning and regrowth strategies, the topological connectivity of the hidden neurons will eventually follow a power law, indicating that the survivability of a neuron is mostly determined by the number of connections it has.*

Furthermore, the topological connectivity of the input layer is also important; therefore, we also studied how the visible neurons form an optimal structure to learn from the training samples. To indicate this, we mapped the topological connectivity of the input layer back onto a heatmap, where the grid has the dimensions of the training samples.

(B) Neuron Decay Secondly, we show that there is a connection between the *Dormant Neuron Phenomena* [48] in Deep Reinforcement Learning (DRL) and sparse neural networks used for image classification. In our experiments, we observe this phenomenon and link it to our contribution A. The number of connections in each layer of the SNN is fixed, meaning there is no increase or decrease in the total number of connections in the network. Combining the fixed sparsity across layers with preferential attachment yields the occurrence of the Dormant Neuron phenomenon.

The Neural Selection Conjecture: The Neural Selection Conjecture posits that in sparse neural networks trained using DST, the network's topology evolves such that the degree distribution of hidden neurons follows a power law. This means that neurons with higher connectivity (i.e., more connections) will continue to gain more connections, becoming stronger over time, while neurons with fewer connections will lose connections, becoming weaker or even dormant. This preferential attachment mechanism is influenced by the regrow strategy employed, where gradient and momentum-based regrow strategies are hypothesized to accelerate this process more aggressively compared to random regrow strategies. Our study aims to empirically validate this conjecture through extensive experimentation and analysis.

To understand why neurons become dormant or even isolated from the artificial neural network, we employ the k-shell decomposition [7]. This method allows us to discover substructures in our network, where neurons are assigned shells. We consider that the shell a neuron is in represents the connectedness of that neuron within the network. The most important substructures we consider are (**i**) the core, (**ii**) the crust, and (**iii**) unimportant neurons. A more detailed description of these substructures can be found in Section 4.5.1.

Lastly, this study will extensively compare regrow strategies and provide insights into the inner workings of sparse neural networks, highlighting the different behaviors observed for each regrow method. Specifically, we found that gradient-based regrow methods tend to create highly connected neurons more rapidly, while momentum-based methods are almost as rapid but slightly more stable. Random regrow methods promote a more uniform distribution of connections.

Our results can be used to design more efficient and effective sparse neural network training procedures. By understanding the distinct effects of different regrow strategies, researchers and practitioners can select the most appropriate method based on their specific application needs. For instance, in scenarios where rapid convergence and the quick formation of highly connected neurons are critical, gradient-based regrow methods might be preferred. Momentum-based methods offer a balance, providing similar rapid connectivity growth with greater stability, which can be beneficial in maintaining performance.

In this study, we relate the efficiency and effectiveness of regrow strategies to network performance. However, in other studies, the requirement might be solely the creation of highly connected neurons at different rates. Therefore, depending on the specific design needs, a choice can be made on how quickly a network needs to achieve a certain structure. Our findings highlight a trade-off: while gradient-based methods quickly create highly connected neurons, there may be benefits in choosing a slightly slower but more stable approach with momentum or random-based methods, depending on the desired outcome.

2 Related Work

Frankle and Carbin [14] introduced the concept known as the Lottery Ticket Hypothesis (LTH), which posits that within a dense neural network, there exist smaller, 'winning tickets'—subnetworks capable of achieving the original network's performance in a more efficient manner. These subnetworks, identified through iterative pruning, not only learn faster but also illuminate the potential for simplifying neural networks without compromising their learning capabilities.

Building upon the idea of network efficiency, Mocanu et al. [39] proposed Sparse Evolutionary Training (SET). Their work demonstrated that introducing topological sparsity to artificial neural networks before training and maintaining it throughout training could drastically reduce the number of parameters needed, thereby reducing the memory footprint and computational power required for the training process without sacrificing performance. In addition to inducing sparsity in ANNs before training, the authors also introduced the first dynamic sparse training algorithm, where connections are pruned based on magnitude and regrown randomly during training. This approach, applicable across various neural network architectures, marked a significant step towards more resource-efficient neural network training.

Furthermore, the literature has seen more state-of-the-art regrow strategies introduced shortly thereafter. Evci et al. [13] proposed gradient regrowth as a novel method to strategically activate connections based on gradient information. Considering gradient-based information for regrowth, Dettmers and Zettlemoyer [11] introduced the momentum regrow strategy, where momentum is defined by the exponentially smoothed gradient, essentially an averaged gradient over time.

Moreover, a comprehensive study performed by Nowak et al. [42] on various pruning criteria within the Dynamic Sparse Training framework showed that magnitude-based pruning tends to be the best choice for pruning highly sparse networks, challenging the pursuit of complexity in pruning criteria. They used the Jaccard index to assess the selection of connections made by various pruning strategies and concluded that magnitude-based pruning methods tend to choose the same selection of connections to prune. In addition, they compared random and gradient regrowth methods; however, no significant impact on performance was found in their analysis.

Gaining more insight into the behavior of neurons in sparse neural networks during training will allow for a better understanding of the impact different regrow methods have on performance. According to Mocanu et al. [39], the hidden neurons in a sparse neural network behave similarly to a scale-free network, where the degree distribution of the hidden neurons exhibits a power law.

Barabási and Albert [6] showed that the exhibition of a power law indicates preferential attachment, but do sparse neural networks follow the same behavior? We observed that, due to the pruning and regrowing of connections, the weights tend to settle (grow strong enough in magnitude to avoid being pruned again) on neurons that already have many connections. One can think of this as the "rich get richer" phenomenon.

Inherent to the "rich get richer" phenomenon is that the "poor get poorer," which in our study means that as strong neurons get stronger, weak neurons will get weaker. The strength is indicated by the number of connections a neuron has. Eventually, this will lead to “dead” neurons in the network. In the context of reinforcement learning (RL), Sokar et al. [48] coined the term Dormant Neuron to describe the "dead" or inactive neurons in an RL agent. Furthermore, they also proposed the *Dormant Neuron Phenomenon*, which is exhibited by a neural network if the number of dormant neurons grows during training. In addition, the authors were able to relate the expressivity and performance of an RL agent to the number of dormant neurons in the artificial neural network, and they improved the performance of the network by reducing the number of dormant neurons.

Method	**Drop**	**Grow**
SET [40]	$\|\theta_-\|, \|\theta_+\|$	random
RigL [13]	$\|\theta\|$	gradient
SNFS [11]	$\|\theta\|$	momentum

Table 1: Overview regrow strategies.

3 Background

Neural networks, inspired by biological neural systems, form the foundation of deep learning and have revolutionized many areas due to their ability to perform a variety of tasks. Originating from the McCulloch-Pitts neuron model in the 1940s, neural networks evolved with the development of the Perceptron in 1958 and significantly advanced with the introduction of the backpropagation algorithm in the 1980s, enabling efficient training of multi-layer networks [38, 43, 44].

At their simplest, neural networks consist of layers of interconnected neurons, where each connection represents a weight that is adjusted during training to minimize a loss function. The inclusion of multiple layers and non-linear activation functions allows these networks to model complex patterns in data far beyond the capabilities of traditional algorithms. Today, neural networks are integral to deep learning, providing powerful tools and applications for analyzing vast datasets and capturing intricate relationships within the data [28].

3.1 Overparametrization

Overparametrization in neural networks, characterized by models having more parameters than training data points, has emerged as a paradoxical trend in deep learning. Contrary to traditional machine learning wisdom that simpler models mitigate overfitting, overparametrized models often excel in both fitting training data and generalizing to new data [57].

The abundance of parameters in overparametrized networks facilitates the optimization process, making it easier for these models to find global minima. The redundancy of parameters contributes to a smoother loss landscape, enhancing the effectiveness of optimization algorithms [24].

Overparametrization also introduces an unexpected regularization effect. The phenomenon where models with excessive capacity still generalize well challenges previous assumptions and is subject to ongoing research. The Lottery Ticket Hypothesis (LTH) is one proposed explanation, suggesting that large networks contain sub-networks configured for optimal performance, which become apparent during training [14].

The benefits of overparametrization come at the cost of higher computational requirements for training and inference, raising concerns about energy use and the environmental impact of deploying large-scale models [51]. The relationship between model size and performance improvement is not linear; beyond a certain point, the gains from additional parameters diminish, indicating a need for careful evaluation of the trade-offs involved [53].

3.2 Sparsification

The challenge of overparametrization in neural networks, characterized by models having more parameters than necessary, can be effectively addressed through sparsification. This approach focuses on minimizing the number of active parameters during both training and inference, aiming to retain only those crucial for the network's performance. Sparsification, applicable to model, ephemeral, and data aspects, will be concentrated here on model and ephemeral sparsification to mitigate computational and memory overheads without significantly compromising accuracy [20].

In the context of **dense** neural networks, fully connected (FC) layers in a multilayer perceptron (MLP) with l layers can lead to approximately N^2 parameters, where N represents the number of neurons per layer. The quadratic increase in parameters underscores the potential benefits of transitioning towards sparsity.

3.2.1 Sparsification techniques

A variety of sparsification techniques exist, each presenting a unique balance between parameter reduction and model performance. Initial methods such as *one-shot pruning*, introduced by Han et al. [18], allowed networks to be pruned a single time after training, reducing parameter count post hoc.

3.2.2 Dense-to-sparse training

Iterative pruning, as explored in subsequent studies [14, 8, 52], demonstrated outcomes by periodically removing a fraction p of weights throughout the training process, with p often diminishing over time. This method has consistently shown that highly sparse models can maintain or even improve performance levels through careful pruning. Two prominent studies, namely LTH and Optimal Brain Damage (OBD), have proven that networks inherently contain sub-networks (winning tickets) that result from efficient iterative pruning [14, 27].

Training and pruning involves techniques such as L_0 regularization, aiming to directly minimize the number of active parameters during training, promoting inherent sparsity within the learning process [34]. In addition to L_0 regularization, which directly aims to minimize the number of non-zero parameters in a model, another related technique that complements sparsity-inducing mechanisms is dropout. Introduced by Srivastava et al. [50], dropout is a form of regularization designed to prevent overfitting by randomly setting a subset of activations to zero at each training step. While not directly a method of sparsity, dropout encourages the model to become less reliant on any single neuron, thus promoting a form of robustness that can align well with sparsity objectives. The combination of dropout with sparsity-inducing methods can lead to more efficient and generalized models.

One-shot pruning methods, such as SNIP and GraSP, are another approach to sparsity. These methods aim to identify and eliminate the least important connections before the training process begins in earnest or very early in training. These methods evaluate the importance of weights based on criteria like sensitivity to the loss function, aiming to prune the network in a way that minimally impacts performance. The appeal of one-shot pruning lies in its simplicity and the immediate reduction in model complexity it offers. However, the challenge is to accurately assess weight importance without the benefit of extensive training. Recent advancements in one-shot pruning techniques continue to refine the criteria for weight importance, aiming to retain the most critical aspects of the model's learning capacity while significantly reducing its size [29, 58].

3.2.3 Sparse-to-sparse training

Sparse-to-sparse training includes both sparse initialization and dynamic sparse connectivity, allowing for adaptive and efficient network architectures that evolve as learning progresses.

Sparse initialization Sparse initialization is a technique where neural networks begin their training with a predetermined sparse structure. This approach involves selectively activating a subset of connections within the network's architecture before any learning occurs, based on predefined criteria or patterns. Mocanu et al. [41] highlighted the use of static sparsity in sparse Boltzmann Machines (SBMs), pointing out the limitations of such a fixed approach, particularly its inability to effectively model well-distributed data due to unchangeable sparse connectivity patterns.

Building on the concept of static sparsity, Mocanu et al. [39] introduced a sparse initialization method, where the connectivity between the layers is based on Erdős-Rényi graphs [12]. Evci et al. [13] later extended this to convolutional neural networks (CNNs), incorporating additional parameters to account for the dimensions of convolutional kernels. An overview of these sparse initialization methods can be found in Table 2.

Dynamic sparse connectivity There are a few disadvantages to having a static connectivity pattern. Firstly, important nodes and connections might be removed before training [14]. This problem can be solved by implementing the concept of dynamic sparse connectivity as proposed by Mocanu et al. [39], whereby topological updates consist of pruning and regrowing connections back in a random manner. Shortly after, Evci et al. [13] proposed a gradient-based regrow method, and Dettmers and Zettlemoyer [11] introduced a momentum-based regrow method. An overview can be found in Table 1.

Sparse initalization	**Description**
Uniform	The sparsity s^l of each individual layer is equal to the total sparsity S.
Erdős-Rényi (ER)[40]	s^l scales with $1 - \frac{n^{l-1}+n^l}{n^{l-1}*n^l}$, where n^l denotes the number of neurons at layer l. This enables sparsity levels to scale relatively to the amount of in- and output features.
Erdős-Rényi-Kernel (ERK) [13]	Scales the number of parameters of the sparse convolutional layers $1 - \frac{n^{l-1}+n^l}{n^{l-1}*n^l*w^l*h^l}$, where w^l and h^l are the width and height of the l'th convolutional kernel, as introduced in the original paper.

Table 2: Overview sparse initialization methods.

4 Methodology

In this section, we establish the theoretical and mathematical foundation for our study. We begin by covering the basics of training dense neural networks and then transition to the process of sparsifying them. Following this, we discuss the technical aspects of sparse neural networks (SNNs), explaining the properties we analyze and the methods we use to test our hypothesis. Our goal is to connect these analysis methods to the fundamental principles of neural networks. By providing a general overview of the problem, we aim to make complex concepts more accessible by gradually breaking them down into digestible parts.

Based on the *Lottery Ticket Hypothesis*, we know that within randomly initialized dense neural networks, there exist subnetworks (winning tickets) that, under certain conditions, can outperform the original dense network. Frankle and Carbin [14] show that these winning tickets have "won" the initialization lottery, meaning the initially sampled weights allow the network to be trained exceptionally efficiently. They achieve this by iterative pruning and resetting the remaining weights to their initial state. Let us split the optimization problem into two parts: optimizing the network parameters and the network topology.

4.1 Dense Training

Given a dense neural network $f(x; \Theta)$, where the parameters Θ include all weights and biases of the network, we optimize these parameters via *backpropagation*, which involves:

1. Computing the gradients of the loss function with respect to each parameter in Θ.
2. Using an optimization algorithm, such as stochastic gradient descent (SGD), to update the parameters based on these gradients.

Considering the cross-entropy loss function over N samples and C classes, where $p(t_c)$ in Eq. 1 represents the probability distribution of the true class labels, and $f_c(\cdot)$ signifies the network's activation for each class, we define the error function for a batch of samples as:

$$E(\Theta) = -\sum_{n=1}^{N}\sum_{c=1}^{C} p_n(t_c) \ln f_c(x_n; \Theta) \tag{1}$$

This error function $E(\Theta)$ can be minimized by updating the parameters Θ using the rule:

$$\Theta^{(t+1)} = \Theta^{(t)} - \eta \sum_{n=1}^{N} \nabla E_n(\Theta^{(t)}) \tag{2}$$

Here, η is the learning rate, t represents the training step, and the gradient ∇E_n represents the derivative of the loss with respect to the parameters, summed over all samples in the batch. These formulations provide a general understanding of how dense neural networks learn, setting the stage for the multi-objective learning task of sparse neural networks.

4.2 Sparse Training

For training sparse neural networks with dynamic sparsity, we integrate sparsity into a traditionally dense neural network $f(x; \Theta)$, which consists of L layers. Sparsity is introduced through a binary mask $M = \{M^{(1)}, \dots, M^{(L)}\}$ applied to each layer of the network, resulting in:

$$f(x; \Theta) = f(x; \Theta \odot M) = f(x; \{\Theta^{(1)} \odot M^{(1)}, \dots, \Theta^{(L)} \odot M^{(L)}\}) \tag{3}$$

In this formulation, $\odot$ represents element-wise multiplication between the parameters $\Theta^{(l)}$ and the mask $M^{(l)}$ at each layer l. The masks $M^{(l)}$ are binary (0 or 1), where 0 denotes a pruned (inactive) connection and 1 denotes an active connection.

We denote the subset of active parameters resulting from the application of the mask as $\theta = \{\theta^{(1)}, \dots, \theta^{(L)}\}$, where $\theta^{(l)} = \Theta^{(l)} \odot M^{(l)}$. The resulting network function is thus defined as:

$$f(x; \theta) = f(x; \Theta \odot M) \tag{4}$$

Here, θ represents the subset of parameters Θ that are active according to the mask M. This subset contains only the parameters corresponding to active connections (where the mask value is 1).

4.3 Mask Updates

By pruning and activating connections between neurons during training, the network can adapt its topology to learn the given task. The total topology of a network is comprised of all its masks. The whole update process can be described in the following steps:

1. Determine the pruning rate $p(t)$ and threshold $\lambda(t)$ at training step t.
2. Prune weights based on predefined criteria.
3. Regrow the same number of weights based on predefined criteria.

4.3.1 Pruning Rate

For each mask $M^{(l)}$ in layer l, it is essential for the adaptation that insignificant connections are pruned. To calculate the pruning threshold with respect to the parameters of the network, we consider a pruning rate $p(t)$ at the start of the training. This pruning rate $p(t)$ decays during training using a cosine function [13]. For example, at the start of training, the pruning rate is set to 50%. This means that 50% of the connections between layers are pruned in the update schedule. As the network learns during training, we reduce the number of connections pruned by decreasing the pruning rate using a cosine function:

$$p(t) = p_0 \left(\frac{1 + \cos\left(\pi \frac{t}{T}\right)}{2} \right) \tag{5}$$

Where p_0 represents the initial pruning rate, which might be, for example, 50%. The variable t denotes the current training step, while T stands for the total number of training steps.

Using this information, we can calculate precisely what the threshold $\lambda(t)$ must be in order to remove the appropriate fraction of the connections. The threshold $\lambda(t)$ is determined such that the fraction of parameters with an absolute value below $\lambda(t)$ equals the pruning rate $p(t)$. Specifically, for each layer l, the threshold $\lambda^{(l)}(t)$ can be found by sorting the absolute values of the parameters in $\Theta^{(l)}$ and selecting the value at the $p(t)$ percentile.

$$\lambda^{(l)}(t) = \text{percentile}\left(\left|\Theta^{(l)}\right|, p(t)\right) \tag{6}$$

Here, percentile $\left(\left|\Theta^{(l)}\right|, p(t)\right)$ denotes the value below which $p(t)$ percent of the absolute values of the parameters $\Theta^{(l)}$ fall. It is important to know that, in this study, the pruning and growing of connections are done in each layer in isolation.

4.3.2 Prune Criteria

There are different strategies that can be used to prune connections between neurons. One of the simplest, yet very effective, methods is pruning based on the absolute value of a weight, also known as magnitude pruning. The magnitude indicates the contribution a connection has on the output of the network. Connections with smaller magnitudes are considered less significant and are pruned first.

The update rule for a mask in layer l, where we use magnitude pruning to update that particular mask, is as follows:

$$M_{ij}^{(l)}(t+1) = \begin{cases} 1 & \text{if } \left|\Theta_{ij}^{(l)}(t)\right| \geq \lambda^{(l)}(t) \\ 0 & \text{if } \left|\Theta_{ij}^{(l)}(t)\right| < \lambda^{(l)}(t) \end{cases} \tag{7}$$

Here,

- $M_{ij}^{(l)}(t+1)$ is the updated mask for the connection between neurons i and j in layer l at time step $t+1$.
- $\Theta_{ij}^{(l)}(t)$ is the weight of the connection between neurons i and j in layer l at time step t.
- $\lambda^{(l)}(t)$ is the pruning threshold for layer l at time step t, calculated as described previously.

This rule ensures that connections with weights whose absolute value is greater than or equal to the threshold $\lambda^{(l)}(t)$ are kept (mask value 1), while those with smaller absolute values are pruned (mask value 0). This method effectively reduces the number of insignificant connections, thus promoting sparsity in the network.

4.3.3 Grow Criteria

Once the pruning step has removed insignificant connections, the network needs to regrow an equivalent number of connections to maintain the same level of connectivity. The number of connections that are regrown is given by a constant ϵ, which matches the number of pruned connections. Here, we describe three different strategies for regrowing connections: random, gradient, and momentum regrowth.

Random Regrowth. New connections are grown by randomly selecting inactive connections (i.e., entries that are zero).

$$M_{ij}^{(l)}(t+1) = \begin{cases} 1 & \text{if } (i,j) \in \text{RandomSelection}\left(\left\{(i,j) \mid M_{ij}^{(l)}(t) = 0\right\}, \epsilon\right) \\ M_{ij}^{(l)}(t) & \text{otherwise} \end{cases} \tag{8}$$

where RandomSelection$(\cdot, \epsilon)$ selects ϵ connections from the total set of inactive connections.

Gradient Regrowth. New connections are grown based on the magnitude of the gradients of the loss function with respect to the weights, favoring connections with higher gradients [13].

$$M_{ij}^{(l)}(t+1) = \begin{cases} 1 & \text{if } (i,j) \in \text{TopK}\left(\left|\frac{\partial E}{\partial \Theta_{ij}^{(l)}}\right|, \epsilon\right) \\ M_{ij}^{(l)}(t) & \text{otherwise} \end{cases} \tag{9}$$

where $\frac{\partial E}{\partial \Theta_{ij}^{(l)}}$ is the gradient of the loss E with respect to the weight $\Theta_{ij}^{(l)}$, and TopK$(\cdot, \epsilon)$ selects the top ϵ connections with the highest gradient magnitudes.

Momentum Regrowth. New connections are grown based on the momentum of the gradients, favoring connections with higher accumulated momentum [11]. The momentum term is defined as a smoothed version of the gradient:

$$v_{ij}^{(l)}(t+1) = \beta v_{ij}^{(l)}(t) + (1-\beta)\frac{\partial E}{\partial \Theta_{ij}^{(l)}}(t) \tag{10}$$

where $v_{ij}^{(l)}(t)$ is the momentum term for the weight $\Theta_{ij}^{(l)}$ at time step t, and β is the momentum coefficient.

The update rule for momentum regrowth is:

$$M_{ij}^{(l)}(t+1) = \begin{cases} 1 & \text{if } (i,j) \in \text{TopK}\left(\left|v_{ij}^{(l)}(t+1)\right|, \epsilon\right) \\ M_{ij}^{(l)}(t) & \text{otherwise} \end{cases} \tag{11}$$

where TopK$(\cdot, \epsilon)$ selects the top ϵ connections with the highest momentum magnitudes.

These growth strategies allow the network to grow connections in an unstructured and dynamic manner. The choice of regrow strategy is a design choice that must be made before the training phase starts, as it directly influences the adaptation process of the mask.

4.4 Introduction to Graph Representation

In this section, we introduce our concept of representing sparse neural networks, particularly sparse MLPs, as directed acyclic graphs. In this representation, nodes (vertices) in the graph correspond to neurons in each layer, and edges represent the connections (weights) between these neurons. This graph-based representation provides a clear and structured way to analyze the topology and connectivity of the neural network.

Formal Definition of a Neural Network A sparse neural network $f(x;\theta)$ (see Eq. 4) can be described as a function $f : \mathbb{R}^n \to \mathbb{R}^m$. The network consists of L layers with k_i neurons in each layer i. Let $L_i = \{1, 2, \ldots, k_i\}$ denote the set of neurons in layer i.

Graph Representation of a Neural Network We can represent the neural network as a directed acyclic graph $G = (V, E, W)$, where V is the set of vertices (neurons), E is the set of directed edges (connections), and W is the set of weights associated with the edges [35, Ch. 44]. Each layer i in the network can be seen as a subset of vertices V_i. An important constraint in this graph representation is that edges can only connect neurons in adjacent layers, ensuring the acyclic nature of the graph.

Formally, the graph representation can be defined as follows:

- Let $V = \bigcup_{i=1}^{L} V_i$ be the set of all vertices in the graph, where V_i represents the set of neurons in layer i and $|V_i| = k_i$.
- Let $E \subseteq \{(v_i, v_j) \mid v_i \in V_l, v_j \in V_{l+1}, 1 \leq l < L\}$ be the set of directed edges representing the connections between neurons in successive layers.
- Let $W = \{\theta_{ij} \mid (v_i, v_j) \in E\}$ be the set of weights associated with the edges, where θ_{ij} is the weight of the connection between neuron v_i in layer l and neuron v_j in layer $l+1$.

Each directed edge $(v_i, v_j) \in E$ corresponds to a non-zero weight θ_{ij} between neuron v_i in layer l and neuron v_j in layer $l+1$. The weight matrix $\Theta^{(l)}$ for layer l can be represented as a sparse matrix, where the non-zero elements correspond to the edges in the graph.

The adjacency matrix $A^{(l)}$ for layer l is defined by the binary mask $M^{(l)}$ applied to the weight matrix $\Theta^{(l)}$, where $M_{ij}^{(l)} = 1$ indicates the presence of a connection (edge) between neurons i and j, and $M_{ij}^{(l)} = 0$ indicates the absence of a connection. Formally:

$$A^{(l)} = M^{(l)} \quad \text{where} \quad A_{ij}^{(l)} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

To incorporate weights, we define the weighted adjacency matrix $W^{(l)}$ for layer l as:

$$W_{ij}^{(l)} = \begin{cases} \theta_{ij} & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

In summary:

$$G = (V, E, W) \quad \text{with} \quad V = \bigcup_{i=1}^{L} V_i \quad \text{and} \quad E \subseteq \bigcup_{l=1}^{L-1} \{(v_i, v_j) \mid v_i \in V_l, v_j \in V_{l+1}\} \tag{14}$$

where $|V_i| = k_i$ for each layer i, and E contains edges representing the non-zero weights between neurons in consecutive layers, as defined by the masks $M^{(l)}$. The set of weights W corresponds to the values of the non-zero elements in the weight matrices $\Theta^{(l)}$.

4.5 Graph Properties

Now we can work with a graph properties of a neural network, we are more interested in the topology of a sparse neural network. Therefore, we do not consider the weights of the graph, only the vertices and edges.

Degree and Degree Distribution The degree of a vertex (neuron) in a graph is the number of edges connected to it. In the context of a neural network, the degree of a neuron corresponds to the number of incoming and outgoing connections it has. Specifically:

- The *in-degree* of a neuron is the number of incoming connections (edges directed towards the neuron).
- The *out-degree* of a neuron is the number of outgoing connections (edges directed away from the neuron).

Considering both the in- and out-degree of a neuron will be referred to as the *degree* of a neuron. The degree distribution of a graph is a probability distribution that describes the fraction of vertices in the graph with a given degree k. Mathematically, the degree distribution $P(k)$ is defined as the fraction of vertices in the graph with degree k. This distribution provides insight into the overall connectivity pattern of the graph. For instance, in a neural network, a few neurons with a high degree might act as hubs, playing a crucial role in information processing—which is also found in brain activity and real-world networks [6, 45, 55].

Examples of Degree Distributions Degree distributions are essential for understanding the topological characteristics of networks. The degree k of a node within a graph G represents the number of connections or edges the node has to other nodes. The degree distribution in an Erdős–Rényi (ER) model, denoted as $G(n, p)$, transitions from a binomial distribution to a Poisson distribution in the limit of large n [25, p. 4]. The probability mass function (PMF) for a degree k in a Poisson distribution is given by:

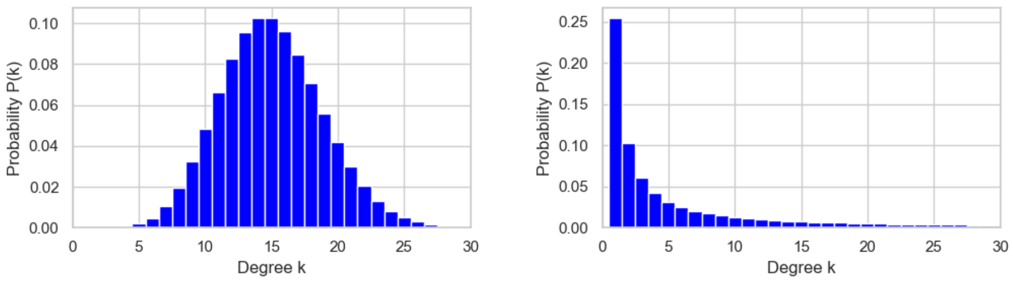
$$P(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{15}$$

where λ is the expected number of occurrences, equating to np for large n, with p being the probability of an edge forming between any two nodes.

Contrary to the ER model, empirical networks often exhibit a degree distribution that follows a power law, particularly in the distribution's (heavy) tail. This is indicative of scale-free networks, which is characterized by a probability density function (PDF):

$$P(k;\alpha) = Ck^{-\alpha}, \tag{16}$$

where α is the scale parameter and C is a normalization constant.



(a) Poisson degree distribution (b) Power-law degree distribution

Figure 1: Plots of the two degree distributions related to this study.

4.5.1 k-shell Decomposition

Exploring beyond simple degree distributions, we turn to k-shell decomposition to uncover substructures within neural networks that influence information flow. The k-shell decomposition (or k-core decomposition) is a method to analyze the structure of a graph by iteratively removing vertices with degrees less than k [7, 46, 4]. The process can be summarized as follows:

- Start with $k = 1$ and remove all vertices with degree less than k.
- Continue to remove vertices with degree less than k until no such vertices remain.
- Increase k and repeat the process.

This technique iteratively peels away the less connected nodes, revealing the k-cores—maximally connected subgraphs where each node has at least degree k. A node's coreness is the highest k for which it belongs to the k-core but not to the $(k+1)$-core [2]. This measure is particularly valuable in assessing the resilience and hierarchical organization of networks, where higher coreness signifies greater centrality and influence within the network's topology [54].

To achieve this decomposition, nodes are recursively pruned if their degree is less than k, with each node's core value signifying the maximum k-core to which it belongs. This recursive process provides insight into the graph's degeneracy, a crucial parameter for understanding network robustness [36]. Recent advancements in algorithmic efficiency have allowed for more scalable analyses, particularly relevant to large and complex networks like those found in bioinformatics and social media studies [5, 1].

Incorporating the information extracted using k-shell decomposition in our study speaks to the flow of information in the network. The nodes in the highest shell (k_{max}) can be regarded as the *backbone* of a network, as illustrated in Figure 10. The k-shell decomposition provides a hierarchical view of the graph's structure, revealing the most connected and central vertices. In the context of neural networks, k-shell decomposition can help identify critical neurons that form the backbone of the network's connectivity.

These graph properties—degree distribution and k-shell decomposition—allow us to quantitatively analyze the connectivity of sparse neural networks. They provide deeper insights into how a sparse neural network evolves during training. Moreover, analyzing sparse neural networks using these properties enables us to measure and compare the effects of different design choices on the overall performance and structure of the network.

4.6 Hypothesis Testing: Power-Law Distributions

To support our claims, we performed rigorous statistical analysis on the degree distributions of neurons in the hidden layers of sparse neural networks. Previous studies, such as Mocanu et al. [39], have shown that the degree distribution of hidden neurons tends to follow a power-law distribution during training. This suggests a *preferential attachment* characteristic, commonly observed in scale-free networks [6].

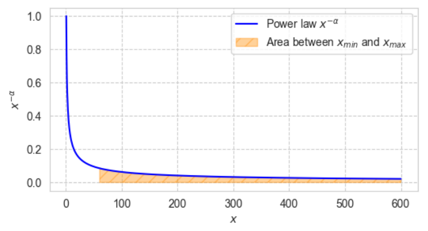


Figure 2: Visualization of the cutoff points $x_{\min}$ and $x_{\max}$ in the fitting procedure of power laws.

In our experiments, we further investigate the degree distribution in the hidden layers using advanced statistical methods to gain deeper insights into the network's behavior. We incorporate statistical significance tests, such as the Kolmogorov-Smirnov (KS) test, and calculate p-values to rigorously evaluate the power-law behavior of the degree distributions over **multiple** points in time during training.

Degrees and Probabilities To analyze the degree distribution of a graph, we first need to convert the graph's structure into a probability distribution. This involves several steps. First, we calculate the degree of each node in the graph, where the degree is defined as the number of edges connected to a node. Next, we count the frequency of each unique degree value observed in the graph. Finally, we convert these frequencies into probabilities by dividing each frequency by the total number of nodes in the graph.

Formally, let k_i be the degree of node i and let n_k be the number of nodes with degree k. The probability $P(k)$ of a node having degree k is given by:

$$P(k) = \frac{n_k}{N} \tag{17}$$

where N is the total number of nodes in the graph. Once we have the probability distribution, we can create the **cumulative distribution (CDF)**. The CDF denoted as $F(k)$ gives the probability that a randomly chosen node has a degree less than or equal to k. It is defined as follows:

$$F(k) = \sum_{j=0}^{k} P(j) \tag{18}$$

where $P(j)$ is the probability of a node having degree j. The CDF is constructed by summing the probabilities for all degrees up to k.

The Kolmogorov-Smirnov (KS) Statistic The KS statistic is used to measure the distance between the empirical cumulative distribution function (ECDF) of the data and the cumulative distribution function (CDF) of the fitted model. It is important to note the difference in notation for the ECDF and CDF, which will be denoted as $\hat{F}(x)$ and $F(x)$, respectively. The KS statistic is defined as follows:

$$D = \sup_{x \geq x_{\min}} \left| \hat{F}(x) - F(x) \right| \tag{19}$$

where D is the KS statistic, sup denotes the supremum (the maximum value) over all x, $\hat{F}(x)$ is the empirical cumulative distribution function of the data for $x \geq x_{\min}$, and $F(x)$ is the cumulative distribution function of the fitted power-law model for $x \geq x_{\min}$.

In words, the KS statistic is the maximum absolute difference between the ECDF and the model CDF over the range of the data that is being fitted (i.e., $x \geq x_{\min}$).

Interpretation of p-values The interpretation of p-values varies with context. Generally, a small p-value in likelihood ratio tests indicates that the observed results are statistically significant and not due to random chance, typically evaluated with a 95% confidence interval ($p < 0.05$). In contrast, for KS tests, a low p-value suggests that the empirical data poorly fits the proposed distribution. A common threshold for fitting power laws is p-value greater than 0.1, which suggests that the power-law model cannot be rejected as a plausible fit for the data [9, p. 3].

Hypothesis Testing Procedure Bringing all the information from this section together, we can define the procedure in order to test our hypothesis:

H_0 (null hypothesis): The degree distribution **is** modeled by a power-law.

H_a (alternative hypothesis): The degree distribution is **not** modeled by a power-law.

Furthermore, we can divide the rest of the process into three parts as follows:

First, we define our probability distribution $\hat{P}(k)$ on which we fit a power-law distribution, resulting in $P(k)$. Next, we calculate the CDFs for both distributions: for our observed data that will be $\hat{F}(x)$ (ECDF) and for the fitted power law $F(x)$. Using both, we can compute the KS statistic $\hat{D}$ (see Eq. 19), which gives the maximum distance between the observed data and the theoretically estimated power law.

In order to assign a significance value to make this result trustworthy, we are going to apply a goodness-of-fit test. Given our fitted power law $P(k)$, we generate N synthetic datasets of size n. We define these synthetic datasets as P_{syn}:

1. **Fit a power-law to (empirical) data and estimate parameters:**
   (a) α: The scaling parameter of the power-law.
   (b) σ: The uncertainty in the estimation of α.
   (c) D (KS statistic): The maximum distance between the ECDF and CDF (see Eq. 19).
2. **Generate N synthetic samples from the fitted power-law:**
   (a) Fit each synthetic dataset to its own power-law distribution. To ensure an unbiased p-value, the KS statistic for each synthetic dataset must be relative to its own best-fit model, not the original distribution.
   (b) Compute the KS statistic ($\hat{D}_{\text{syn}}$) for each synthetic dataset with its **own** fitted model.
3. **Compute the p-value:**

$$p = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{D}^i_{\text{syn}} > D) \tag{20}$$

   where $\mathbb{I}(\cdot)$ is the indicator function that equals 1 if the condition is true and 0 otherwise, and $\hat{D}^i_{\text{syn}}$ is the KS statistic for the i-th synthetic dataset.

This hypothesis testing procedure is conducted over multiple points—not just the start and end of training—in time during training, because we are interested in the process of how the topology of a network evolves and whether it evolves towards a power-law distribution.

5 Experiments

5.1 Experimental Setup

In the experiments, we use a batch size of 128, employing the SGD optimization algorithm with a momentum of 0.9. A weight decay of 0.0005 is applied, alongside Nesterov momentum to enhance convergence. We employ a *multi-step* learning rate. The learning rate starts at 0.1 and is decreased at two points in time: at 50% epochs (halfway) to 0.01 and once more at 75% epochs to 0.001. This multi-step learning rate is beneficial for overcoming local minima (see Figure 3). Furthermore, we use ERK as the sparse initialization method to achieve a total sparsity of 95% across the network. These sparsity levels are normalized across layers, indicating that sparsity is not uniformly distributed among them. The sparsity is normalized by increasing the sparsity of "bigger" layers and decreasing the sparsity of "smaller" layers, weighing them by their size. The initial pruning rate targets 50% of the sparse parameters, which then gradually decays using a cosine annealing schedule.

We keep the sparsity levels fixed across layers, so connections are not redistributed across layers during the exploration of the sparse topology. Every 400 batches (Δt) we explore sparse connectivity. Converting this to samples - 128 samples per batch - results in $400 \cdot 128 = 51,200$ samples, leading to topological changes every $\sim$1.14 epochs for CIFAR10 and CIFAR100; $\sim$0.95 epochs for MNIST and Fashion MNIST.

In Table 5, there is an overview of some simpler models used in this study. Models often used in the literature that are not provided in the table, but used in this study, include the original ResNet-50 and ResNet-56 models, which have 25,502,912 and 855,770 parameters, respectively [19]. Additionally, we consider VGG-16 [47], with its implementation based on [33] (as mentioned in [42]). An overview of the datasets used in the experiments can be found in Table 6.

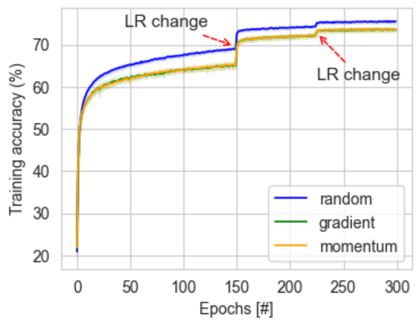


Figure 3: The effect of the multi-step learning rate while training CNNs. Red arrows indicate the points of decrease in learning rate.

5.2 Results

Table 4 presents test accuracies for the highly sparse (95%) models trained with either random, gradient, or momentum regrowth. We also provide a dense baseline for each model for comparison. The characteristics of the used datasets can be found in Table 3.

Model/Dataset	**CIFAR10**	**CIFAR100**	**MNIST**	**Fashion MNIST**
ConvNet	85.88 ± 0.36	59.68 ± 0.33	99.32 ± 0.11	91.95 ± 0.26
MLP	63.51 ± 0.27	29.57 ± 0.32	97.67 ± 0.1	88.22 ± 0.18
ResNet-50	93.75 ± 0.24	72.41 ± 0.24	99.59 ± 0.02	93.46 ± 0.19
ResNet-56	93.91 ± 0.25	72.71 ± 0.23	99.6 ± 0.02	93.44 ± 0.16
VGG-16	93.78 ± 0.14	74.63 ± 0.25	99.59 ± 0.03	93.38 ± 0.12

Table 3: Test results for the models trained with full density, averaged over 5 seeds.

Models	Method	CIFAR10	CIFAR100	MNIST	FashionMNIST
Sparsity 0.95					
ConvNet	random	**77.12 ± 0.30**	33.40 ± 18.12	**99.17 ± 0.09**	**90.37 ± 0.33**
	gradient	63.50 ± 12.93	**38.09 ± 0.93**	43.81 ± 32.01	69.10 ± 28.81
	momentum	74.12 ± 4.85	38.07 ± 1.58	62.14 ± 26.81	89.79 ± 0.21
MLP	random	**68.08 ± 0.07**	**37.13 ± 0.27**	96.13 ± 0.28	**86.26 ± 0.37**
	gradient	65.83 ± 0.43	34.61 ± 0.22	96.13 ± 0.15	79.50 ± 4.08
	momentum	66.25 ± 0.31	35.14 ± 0.36	**96.26 ± 0.28**	85.64 ± 0.41
ResNet-50	random	**91.24 ± 0.21**	**66.48 ± 0.37**	99.55 ± 0.06	**93.40 ± 0.15**
	gradient	91.07 ± 0.08	64.84 ± 0.72	**99.59 ± 0.06**	93.34 ± 0.08
	momentum	90.8 ± 0.39	64.58 ± 0.56	99.55 ± 0.07	93.16 ± 0.19
ResNet-56	random	**91.45 ± 0.18**	**66.9 ± 0.35**	99.59 ± 0.05	**93.42 ± 0.14**
	gradient	91.40 ± 0.13	65.41 ± 0.43	**99.55 ± 0.04**	93.30 ± 0.20
	momentum	91.21 ± 0.26	65.24 ± 0.55	99.55 ± 0.03	93.34 ± 0.25
VGG-16	random	**93.07 ± 0.17**	72.54 ± 0.15	**99.58 ± 0.04**	93.49 ± 0.11
	gradient	92.95 ± 0.21	72.75 ± 0.45	99.51 ± 0.04	**93.50 ± 0.22**
	momentum	93.06 ± 0.34	**72.77 ± 0.44**	99.56 ± 0.04	93.29 ± 0.16

Table 4: Test Performance of different models and methods on various datasets (5 seeds)

5.3 (A) Topology evolution

To gain a sophisticated insight into the network's topological composition and how various regrow strategies affect this structure, we train a sparse MLP for 300 epochs on the CIFAR10 dataset using the hyperparameters as specified in Section 5. The network's states are saved before, during, and after training. During training, the network's state is saved after every Δt (topological update). Note: the distribution of connections between **hidden** neurons.

Topological connectivity We trained MLPs over 5 runs using magnitude pruning combined with random, gradient, and momentum regrowth. All the models are initialized in the same manner, namely, with the ERK method. Figure 4 shows the pre-training distribution of each model.

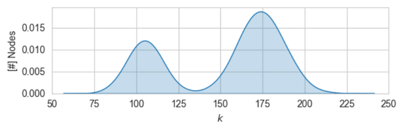


Figure 4: Initial degree distribution, that is, after sparse initialization (ERK) and pre-training.

In Figure 5, the distributions of connections among neurons and where the distribution ends are shown. We observe that eventually all the models' distributions move towards a power law. Moreover, gradient and momentum regrowth tend to speed up this behavior, hence, converge faster. In simple terms, the connectivity changes among neurons of the models trained with gradient and momentum regrowth are more significant when compared with random regrowth. The striped lines in Figure 5 indicate the end of a particular distribution. Interestingly, after a few topological updates, gradient and momentum regrowth have already drastically changed the topological structure. In other words, the variance (spread) of these distributions has increased significantly in a short amount of time. This resulted in a structure where a few nodes have become very strong (high degree) and some very weak (low degree). We assume that the strength of a neuron is determined by its activation, which is directly influenced by the number of connections and their magnitude. In contrast, models trained with random regrowth slowly increase the variance of the distribution. Where the variance of random regrowth distributions keeps increasing during training, gradient and momentum increase and decrease, aiming at a more "aggressive" search of the connectivity landscape.

The Neural Selection Hypothesis. Our null hypothesis states that the degree distribution **is** sampled from a power law. We reject the null hypothesis if $p < 0.95$. By rejecting the null hypothesis, we assume the alternative to be true, that is, the degree distribution of the hidden neurons is **not** sampled from a power law.

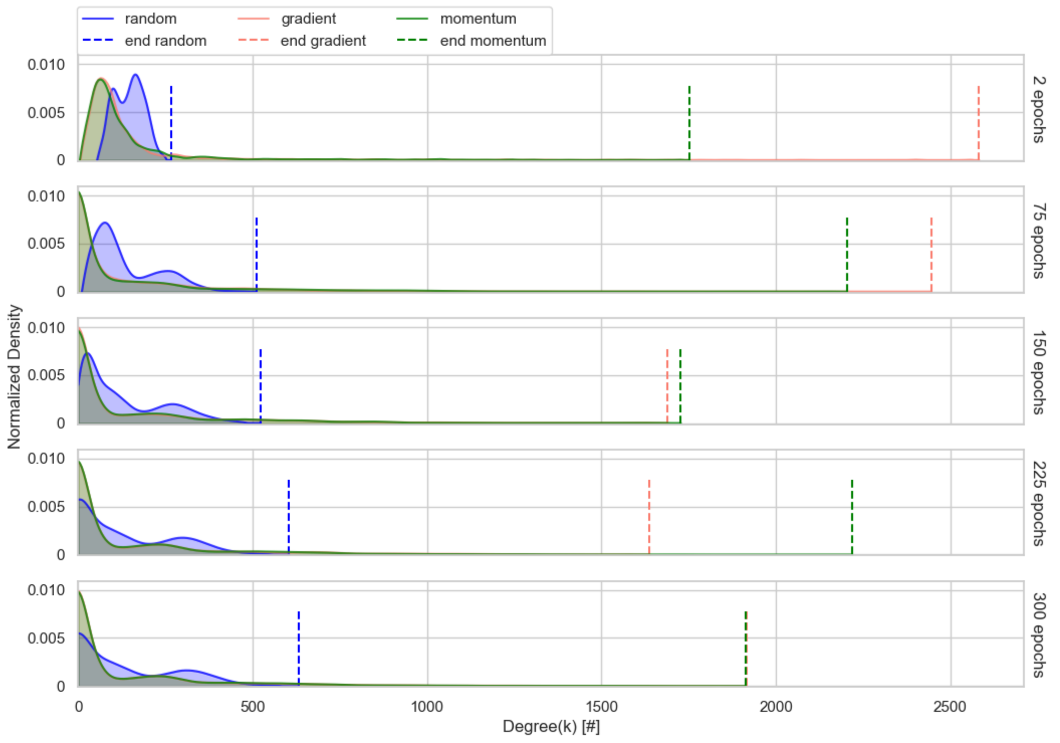


Figure 5: Degree distribution of the sparse MLPs during the learning process (averaged over 5 seeds). Comparing random, gradient, and momentum regrowth. The solid lines are the distributions and the striped lines indicate where each distribution ends. The rows indicate different consecutive points in time taken at 2, 75, 150, 225, and 300 epochs, respectively.

Fitting power laws to our empirical data is a complex process; therefore, we use a framework specifically designed for this [3]. This framework estimates the scalar component (α) from a power law using Maximum Likelihood Estimation. Estimating this scalar component can also contain some uncertainty σ. For each regrow strategy, we plot the most important parameters, according to Section 4.6, to show the difference in behavior.

Firstly, considering all plots, we can see the parameters stabilize over time for every regrow method. Secondly, we consider the scalar component α to be in line with scale-free networks [6], as they often have a scalar component where $2 < \alpha < 3$; the estimation improves during training since the error (σ) decreases. Thirdly, the maximum distance between our empirical data and the fitted power law (D) also decreases. Lastly, the p values for gradient and momentum vastly increase at the beginning of training. This is contrary to random, which slowly increases.

Based on our experiments, we consider random regrowth to converge slower than gradient and momentum. When we ran the same experiment for random with lower Δt, resulting in more topological updates, random also reached over the significance threshold.

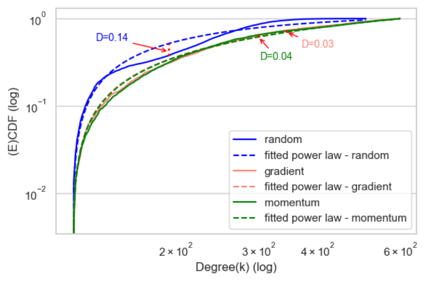


Figure 6: Visualization of how the D value from the KS-test is calculated, applied to our empirical data.

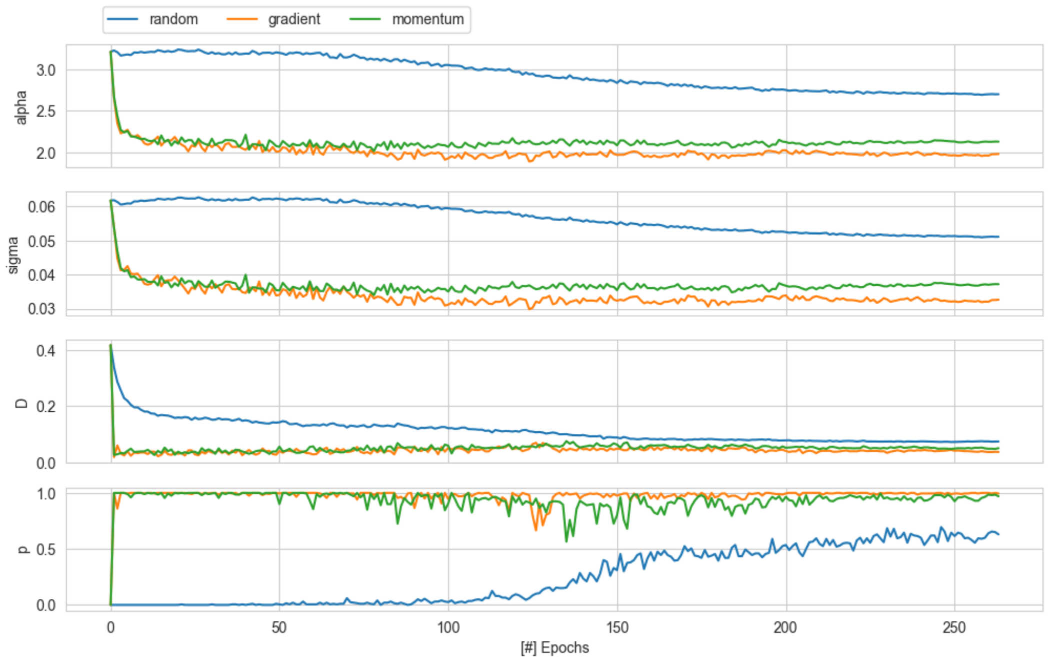


Figure 7: Visualizing the behavior of the parameters (Sec. 4.6) considered when testing the hypothesis per regrow strategy, averaged over 5 seeds on the CIFAR10 dataset, using MLP architecture.

In general, we can consider that gradient and momentum regrowth have similar effects on the topological properties of trained networks. Random tends to 'converge' slower based on Figure 7. We can confirm that gradient and momentum regrowth behave in a very similar manner. After the first topological update.

Connectivity pattern input Another interesting discovery is how the sparse input layer of our MLPs maps back to the input features of the training images and how the first hidden layer behaves according to that.

The formation of the visible neurons trained with random regrowth is much more symmetric, homogeneous, and spread out. For gradient and momentum, this is not the case. It seems that gradient and momentum regrowth, again, make some neurons very strong (i.e., high degree), thereby also concentrating their attention on a very small part of the image. In contrast, random regrowth spreads the attention over the image. Examples of the corresponding training samples for the CIFAR10 dataset can be found in Appendix F.

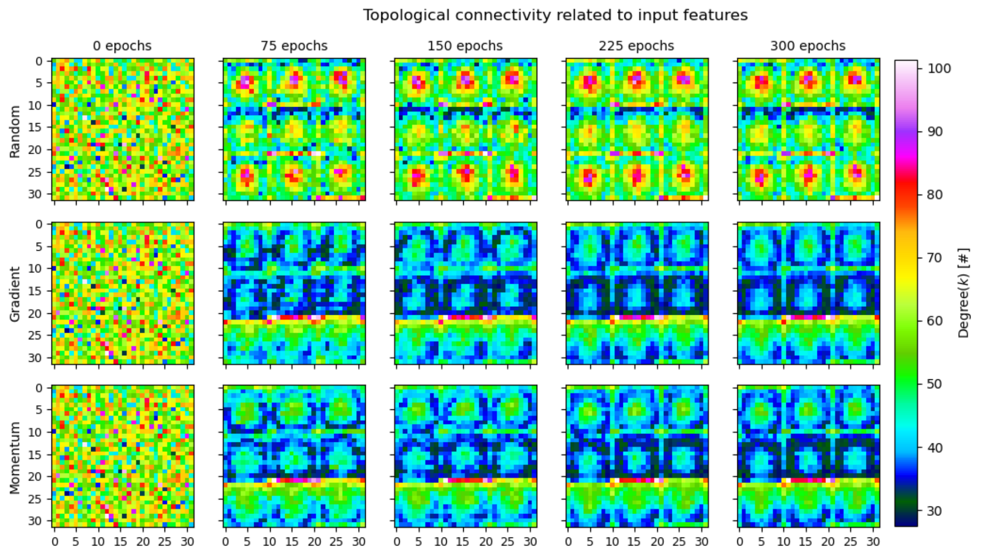
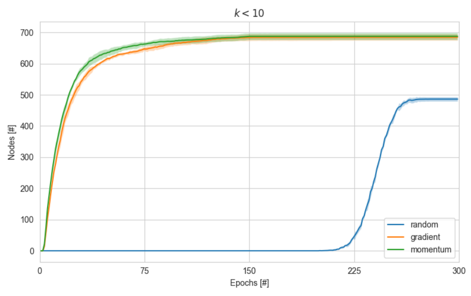


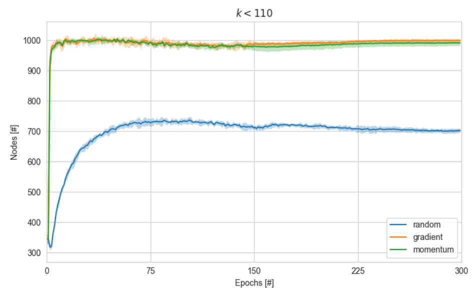
Figure 8: Heatmap visualization of the sparse **input layer** mapped back to the input features of an image (averaged over 5 seeds). The rows separate regrow methods random, gradient, and momentum, respectively. The columns indicate the state at a specific point during training (epochs). The x- and y-axes of the subplots indicate the pixels.

5.4 (B) Neuron decay

In Figure 9, we present two plots showing the increase of dormant neurons during the training of our models. As in previous experiments, the models trained in this experiment are sparse MLPs with a sparsity of 95%, using state-of-the-art regrow strategies. For this experiment, we considered two different thresholds: 10 and 110. In Figure 9a, we assumed—keeping in mind the degree distributions—that 10 would be a suitable threshold. In Figure 9b, we used a threshold of 110. This particular threshold is the same threshold used for fitting the power law distributions and was used as the x_{min} parameter in the fitting process (Sec. 4.6).

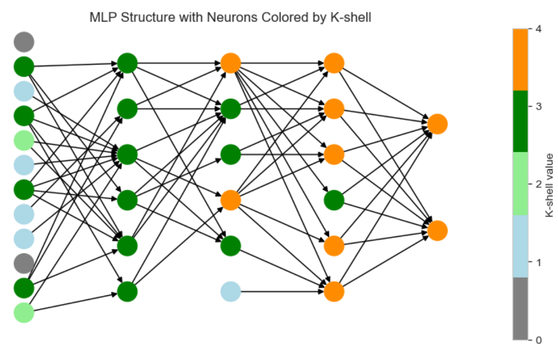


(a) Dormant neuron count, where $10 > k$

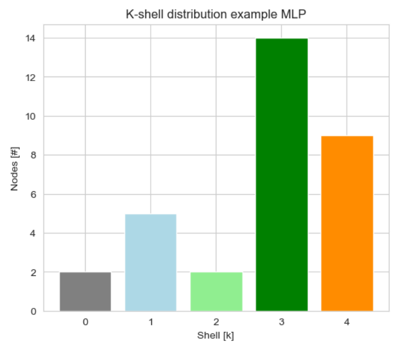


(b) Dormant neuron count, where $110 > k$

Figure 9: Increase in dormant neurons during training across different regrow strategies. Given different thresholds (a) 10 and (b) 110, compared with the degree k (number of connections) of a neuron. Measurements are performed over 5 seeds, per regrow method.



(a) Example of sparse MLP to visualize the shells of nodes.



(b) Bar chart of the k-shell distribution for the nodes in the example MLP.

Figure 10: Visualization of nodes in different shells resulting from the k-shell decomposition given an example MLP with an arbitrary architecture.

We observe in Fig. 9 that gradient and momentum regrowth are more aggressive in creating dormant neurons compared to random. Using random regrowth will only result in more dormant neurons much later in the training process. Interestingly, in both Figure 9a and Figure 9b, we observe that independent of the regrow method, the number of dormant neurons tends to stabilize over time.

K-shell decomposition Given that the structure of a sparse neural network can be represented by a graph structure, we use k-shell decomposition [7] to analyze the connectedness of neurons throughout the network, as our architecture consists of multiple sparse layers.

To make this easier to understand, consider the analogy of peeling an onion. It consists of multiple layers and contains a core. The further one peels, the closer one gets to the core, and eventually, when there are no layers left to peel, the core is reached. In our case, the core of the onion is the k_{max}, which represents the highest shell of the network. All the layers around the core, that is, shells from $k_{max} - 1$ to k_{min}, we consider to be the crust.

We have already seen in Section 5.3 that the distribution of degrees among the neurons is highly skewed. Looking at Figure 12, we observe that at the end of every distribution, there is an accumulation of neurons that represent the core of the networks; these can be considered the most important neurons. We assume that the neurons in the core are best connected, have more activity, and are more likely to keep their connections together while growing new connections. This ensures they are not pruned in the next cycle and have a better chance of surviving and accumulating new connections. These neurons are also updated more frequently during *backpropagation*.

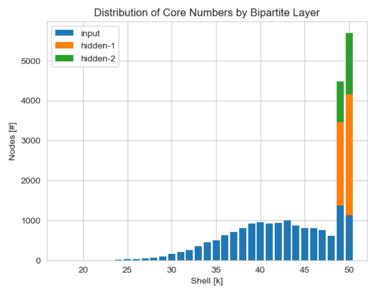


Figure 11: K-shell distribution before training.

Before training, all the models start with the same topological structure. Therefore, we plot their k-shell distribution in Figure 11. During training, we see a noticeable difference in behavior in Figure 12. Again, gradient and momentum are better strategies for creating denser backbone structures, which directly results in some neurons becoming dormant or even isolated.

6 Discussion

Looking at Table 4, we see that random regrowth generally leads to better performance in our experiments. However, this contradicts other studies [13, 11, 21] that introduce different sparse training algorithms. The reason for this could be insufficient hyperparameter optimization. Each algorithm is proposed with a different set of recommended hyperparameters, although they are tested on the same dataset(s). In this study, we used the same hyperparameters for all regrow strategies to allow for a fair assessment of solely the regrow strategies.

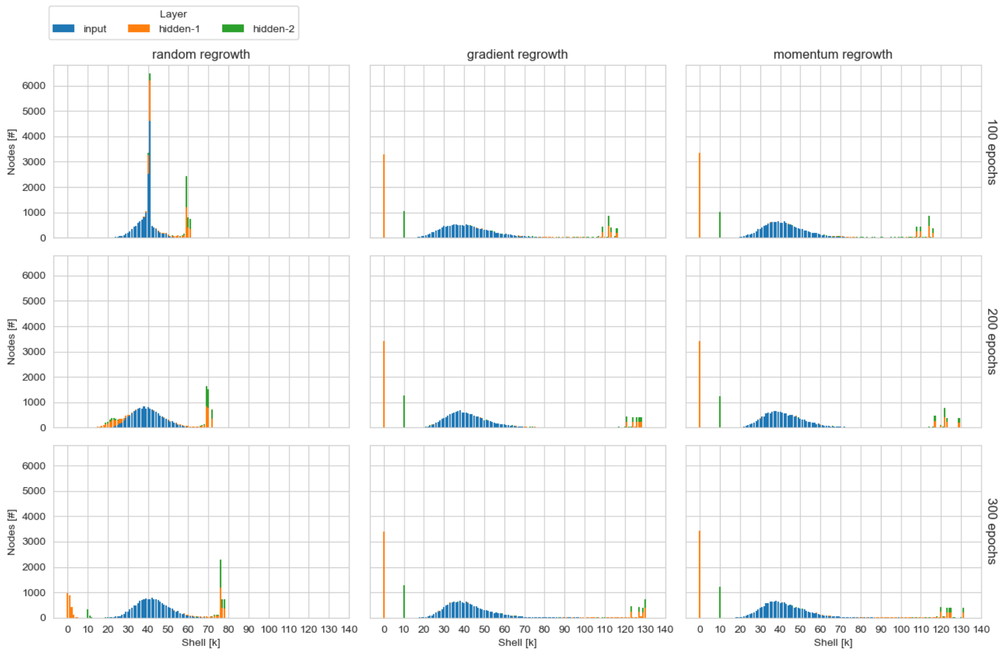


Figure 12: Distributions of the counted shells for the visible (**blue**) and hidden (*green and orange*) neurons in our trained MLPs with different regrow strategies. It's important to note that on the x-axis for hidden layer 1 and 2, the **isolated neurons** (neurons in k_{min}) are indicated at the **0** and **10** mark, respectively. For the second layer, the minimum shell is 10 due to the last layer being fully connected.

Our claim of the Neural Selection Hypothesis aligns with the study performed by Mocanu et al. [39]. The conjecture proposed in this study is also in line with the Dormant Neuron phenomenon in the field of DRL proposed by Sokar et al. [48]. While analyzing the behavior of the hidden neurons in our experiments, we noticed something unexpected. We did not anticipate the magnitude of difference between random and gradient-based regrow strategies. In other words, the rate at which highly connected neurons are assembled is substantial.

Furthermore, the degree distributions of the input layer differ when compared to the hidden layers, as shown in Appendix G. Additionally, in Figure 8, we show that there is a difference in how the network distributes its attention when choosing different regrow strategies. We see that for random regrowth, while the network is being trained, the attention is spread more symmetrically and homogeneously when mapped back to the dimensions of the training samples. However, this is not the case for gradient and momentum regrowth. The attention seems to be focused very densely on smaller parts of the image.

7 Conclusion

(A) The Neural Selection Hypothesis. Based on our empirical results, we provide statistical evidence that the connections of hidden neurons will eventually follow a power law, regardless of the chosen regrow strategy. However, the choice of regrow strategy influences the speed at which the distribution changes during training. Gradient and momentum regrowth are more aggressive in creating highly connected neurons compared to random regrowth, which activates connections more uniformly.

(B) Neuron Decay. We demonstrate that the power law indicates a preferential attachment mechanism in high sparsity regimes. During training, the topological structure of sparse neural networks changes, with connections of weaker neurons being pruned and re-established at stronger neurons. This behavior is driven by the higher activation of stronger neurons, leading to larger gradient updates during backpropagation. Consequently, neurons with fewer connections receive smaller updates, hindering their ability to increase the magnitude of newly grown connections above the pruning threshold.

General. Overall, based on our experiments, there are a few key conclusions that generally apply to this study. Comparing Table 3 and Table 1, we observe that sparse MLPs particularly benefit from inducing sparsity. They are able to perform similarly to or even outperform their dense counterparts on all datasets, regardless of the chosen regrow strategy, with a sparsity level of 95%. Furthermore, the performance differences between regrow strategies are minor. However, random regrowth generally leads to better performance in most cases. If the goal is to create many hubs (well-connected neurons) regardless of the network's performance, one could employ gradient or momentum regrowth due to their ability to enforce the preferential attachment mechanism more aggressively.

8 Limitations & Future Work

In this section, we discuss the limitations of this study and what can be gained by overcoming them. To start, a crucial part of the sparse-to-sparse training method is the update scheduler. This update scheduler is the phase where the topological connectivity of a network is explored by pruning and regrowing connections. However, in this study, we used a fixed interval for the update scheduler. In other words, regardless of the choice of the regrow strategy, the interval of the update scheduler remained fixed. This also applies to the decay schedule of the pruning rate, for which we chose a cosine decay function (Eq. 5) [13]. Since gradient and momentum regrowth are very quick in creating hubs, it would be worthwhile to investigate how sensitive the correlation between the pruning rate and the creation of dense hubs is for each regrow strategy.

Furthermore, extending the analysis we performed in this study to other architectures, including the MLP, will provide more evidence for a general conclusion. For example, CNNs are often used to solve computer vision tasks; therefore, using the same analysis on CNNs could provide insight into their behavior and inner workings.

Moreover, the analysis on the behavioral component of this study aims to prove that the connections of a network tend to establish themselves throughout training to neurons that already have many connections. The distribution that correlates with this preferential attachment mechanism is a power law. In this study, we discuss a power-law distribution as if it is only one distribution, but based on [9], it would be more useful to consider that a power law is a family of distributions, which include the log-normal, truncated power law, and exponential distribution. Keeping in mind that we only hypothesized the degree distributions of networks to behave like a power-law distribution (defined as Eq. 15), a more sophisticated and well-defined approach would be to employ likelihood ratio tests to test for all the members of the power law family.

On a different note, we noticed some instability when using gradient regrowth. Different studies have researched the effect of batch size and the use of (stochastic) gradient descent to update a neural network's parameters [37, 30, 22]. The general conclusion based on these works is that a low batch size in combination with SGD leads to unstable performance when training artificial neural networks. Considering DST and the instability that arises with too many gradient updates, we intuitively link this to the update scheduler, that is, the pruning and regrowing of connections. For example, a possible explanation for the instability in models trained with gradient regrowth could be due to the parameters being updated with a gradient-based method, namely, SGD, while the topology is also updated using a gradient-based method, in our case, choosing a gradient-based regrowth method in the update scheduler.

Moreover, theories concerning preferential attachment typically assume that the network experiences a growing population, as observed in real-world networks like the internet [6]. It would be interesting to examine where the connections of neurons tend to stabilize when the population is increasing, particularly when the total number of degrees in a layer is not fixed. This raises the question: does the Neural Selection Hypothesis hold when an artificial neural network has layers that can grow and shrink? To clarify, the given condition is that the number of connections for each layer in the network is not fixed.

Another aspect worth highlighting is the exploration of the topological space of the artificial neural networks during the training phase. Liu et al. [32] address this problem and claim that performance is highly related to the total number of reliably explored parameters throughout training. They consider a parameter reliably explored if the connection is newly grown and the connection has grown enough in magnitude to not fall below the pruning threshold for some time. Applying this metric could possibly provide a clearer insight into the distinction between regrow methods and show which method is more effective in exploring the topological space.

To make a stronger claim on the mechanism of preferential attachment, one could also track the activation of neurons during training. The number of connections of a neuron is only half of the story; their weights also matter. Adding an analysis where the weights are also considered and used in explaining the preferential attachment mechanism could ensure a more complete understanding of why the distribution behaves like a power law. It would be interesting to know how the distribution of connections is amongst winning lottery tickets. Technically, this can be done by considering a weighted graph as shown in Eq. 14.

References

[1] k-core decomposition. Retrieved from TigerGraph Graph Data Science Library: `https://docs.tigergraph.com/dev/gsql-ref/querying/operators-and-functions#k-core-decomposition`.

[2] k-core. 2022. Retrieved from Wikipedia: `https://en.wikipedia.org/wiki/K-core`.

[3] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. powerlaw: A python package for analysis of heavy-tailed distributions. *PLoS ONE*, 9(1):e85777, January 2014. ISSN 1932-6203. doi: 10.1371/journal.pone.0085777. URL `http://dx.doi.org/10.1371/journal.pone.0085777`.

[4] J Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in neural information processing systems*, pages 41–50, 2006.

[5] Gary D Bader and Christopher WV Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, 2003.

[6] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286 (5439):509–512, October 1999. ISSN 1095-9203. doi: 10.1126/science.286.5439.509. URL `http://dx.doi.org/10.1126/science.286.5439.509`.

[7] Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt, and Eran Shir. A model of internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences of the United States of America*, 104:11150–4, 08 2007. doi: 10.1073/pnas.0701175104.

[8] Jingfei Chang, Yang Lu, Ping Xue, Yiqun Xu, and Zhen Wei. Iterative clustering pruning for convolutional neural networks. *Knowledge-Based Systems*, 265:110386, 2023.

[9] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, November 2009. ISSN 1095-7200. doi: 10.1137/070710111. URL `http://dx.doi.org/10.1137/070710111`.

[10] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[11] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[12] A. Erdős, P. & Rényi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959.

[13] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the Lottery: Making All Tickets Winners, July 2021. URL `http://arxiv.org/abs/1911.11134`. arXiv:1911.11134 [cs, stat].

[14] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, March 2019. URL `http://arxiv.org/abs/1803.03635`. arXiv:1803.03635 [cs].

[15] David Freedman, Robert Pisani, and Roger Purves. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[16] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[17] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference: revised and expanded*. CRC press, 2014.

[18] 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.

[19] 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.

[20] 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, 2021.

[21] Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *CoRR*, abs/2106.03517, 2021. URL `https://arxiv.org/abs/2106.03517`.

[22] Satyen Kale, Ayush Sekhari, and Karthik Sridharan. Sgd: The role of implicit regularization, batch-size and multiple-epochs, 2021.

[23] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[24] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.

[25] Yi-Xiu Kong, Gui-Yuan Shi, Rui-Jie Wu, and Yi-Cheng Zhang. k-core: Theories and applications. *Physics Reports*, 832:1–32, 2019. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2019.10.004. URL `https://www.sciencedirect.com/science/article/pii/S037015731930328X`. k-core: Theories and Applications.

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[27] Yann Lecun, John Denker, and Sara Solla. Optimal brain damage. volume 2, pages 598–605, 01 1989.

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[29] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[30] Deren Lei, Zichen Sun, Yijun Xiao, and William Yang Wang. Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications, 2018.

[31] 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.

[32] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training, 2021.

[33] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. *arXiv preprint arXiv:2202.02643*, 2022.

[34] Christos Louizos, Max Welling, and D Kingma. Learning sparse neural networks through. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*.

[35] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[36] Fragkiskos D Malliaros, Matteo-Per Rossi, and Michalis Vazirgiannis. The k-core decomposition: a survey. *EPJ Data Science*, 9(1):16, 2020.

[37] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

[38] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[39] 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):2383, June 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04316-3. URL `https://www.nature.com/articles/s41467-018-04316-3`. Number: 1 Publisher: Nature Publishing Group.

[40] 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, March 2021. URL `http://arxiv.org/abs/2103.01636`. arXiv:2103.01636 [cs].

[41] Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Wil L Kling. Deep learning for estimating building energy consumption. *Sustainable Energy, Grids and Networks*, 6:91–99, 2016.

[42] Aleksandra I. Nowak, Bram Grooten, Decebal Constantin Mocanu, and Jacek Tabor. Fantastic weights and how to find them: Where to prune in dynamic sparse training, 2023.

[43] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[44] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[45] Majid Saberi, Reza Khosrowabadi, Ali Khatibi, Bratislav Misic, and Gholamreza Jafari. Topological impact of negative links on the stability of resting-state brain network. *Scientific reports*, 11(1):2176, 2021.

[46] Stephen B Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. *arXiv preprint arXiv:2302.12902*, 2023.

[49] Charles Spearman. Correlation calculated from faulty data. *British journal of psychology*, 3(3):271, 1910.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

[51] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

[52] Chong Min John Tan and Mehul Motani. DropNet: Reducing neural network complexity via iterative pruning. 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 9356–9366. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/tan20a.html`.

[53] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The computational limits of deep learning, 2022.

[54] Rohit Tripathy, Kamal Kanth, and Khaled Salem. Scalable k-core decomposition for static graphs using a dynamic graph data structure. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 9–18. IEEE, 2018.

[55] Martijn P Van den Heuvel and Olaf Sporns. Network hubs in the human brain. *Trends in cognitive sciences*, 17(12):683–696, 2013.

[56] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[57] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2017.

[58] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W. Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training, 2021.

A Exploratory experiments

Based on previous research [42, 11], we decided to perform some exploratory experiments to confirm which pruning method was most stable and seemed to work best in high sparse regimes. Nowak et al. [42] concluded that the best performing pruning method in high sparse regimes - also the most simple - is magnitude pruning. To confirm this we employ basic magnitude pruning and the magnitude pruning used in SET. Both are magnitude based pruning methods, the difference is that basic magnitude pruning takes the absolute weights and SET considers both positive and negative weight separately, as shown in Table 1. The results of our preliminary pruning comparison can be found in 13. Based on those results we can conclude that simple magnitude pruning performs best overall with each regrow strategy. Interestingly, it appears that using random strategy for pruning and regrowth yield the worst results consistently. Therefore, we recommend using magnitude or gradient information criteria for at least one of the topology mutation steps (pruning or regrowth).

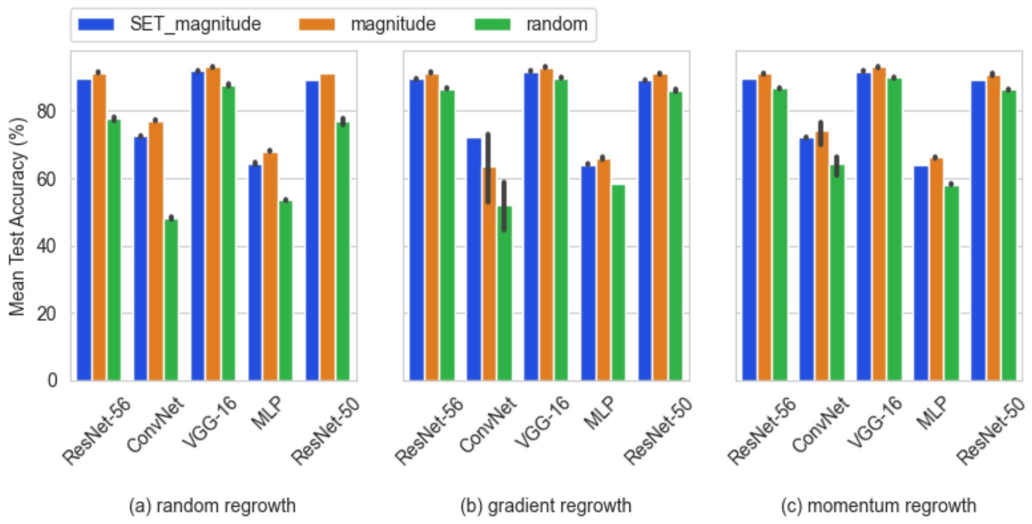


Figure 13: Comparing **pruning** methods combined with regrow strategies and model architectures on the CIFAR10 dataset.

B Experimental setup

In this Section we give details regarding the architectures (Tab. 5) and the datasets (Tab. 6).

Layer	MLP	ConvNet
1	Linear(3 × 32 × 32, 1024)	Conv2d(3, 32, 3, 1)
2	ReLU	ReLU
3	Linear(1024, 512)	MaxPool, size 2
4	ReLU	Conv2d(32, 64, 3, 1)
5	Linear(512, 10)	ReLU
6		MaxPool, size 2
7		Conv2d(64, 128, 3, 1)
8		ReLU
9		Global Average Pool
10		Linear(128, 10)
Num. Params (density)		
100% (Dense)	3,676,682	94,538
5% (95% Sparse)	183,834	4,727

Table 5: The MLP and CNN architectures used in this study.

Dataset	Features	Classes	Samples	Train	Validation	Test
CIFAR10 [26]	3x32x32	10	70,000	45,000	5,000	10,000
CIFAR100 [26]	3x32x32	100	70,000	45,000	5,000	10,000
dataset split			100%	75%	8%	17%
FashionMNIST [56]	1x28x28	10	70,000	53,900	6,300	9,800
MNIST [10]	1x28x28	10	70,000	53,900	6,300	9,800
dataset split			100%	77%	9%	14%

Table 6: The characteristics of the datasets used in the experiments.

C Experiments with more sparsity levels

Models	Method	CIFAR10	CIFAR100	MNIST	FashionMNIST
Sparsity 0.90					
ConvNet	random	**81.07** ± **0.11**	**49.94** ± **0.46**	**99.26** ± **0.10**	**90.86** ± **0.13**
	gradient	68.84 ± 15.77	47.74 ± 0.56	51.53 ± 43.63	90.67 ± 0.08
	momentum	80.67 ± 0.56	47.97 ± 0.43	71.18 ± 38.08	90.75 ± 0.15
MLP	random	**68.17** ± **0.33**	**36.89** ± **0.35**	97.14 ± 0.18	**87.56** ± **0.29**
	gradient	66.29 ± 0.13	35.15 ± 0.31	**97.15** ± **0.12**	86.97 ± 0.52
	momentum	66.64 ± 0.30	35.52 ± 0.3	97.14 ± 0.16	87.33 ± 0.07
ResNet-50	random	**92.15** ± **0.19**	**69.74** ± **0.20**	99.58 ± 0.04	93.57 ± 0.15
	gradient	92.06 ± 0.21	68.95 ± 0.32	99.52 ± 0.08	93.48 ± 0.22
	momentum	92.02 ± 0.3	68.98 ± 0.33	**99.59** ± **0.03**	**93.57** ± **0.10**
ResNet-56	random	**92.30** ± **0.36**	**69.99** ± **0.29**	**99.60** ± **0.05**	93.53 ± 0.26
	gradient	92.25 ± 0.25	69.51 ± 0.16	99.54 ± 0.05	93.45 ± 0.14
	momentum	92.09 ± 0.47	69.24 ± 0.32	99.56 ± 0.05	**93.61** ± **0.28**
VGG-16	random	**93.50** ± **0.11**	73.70 ± 0.14	99.52 ± 0.06	**93.50** ± **0.11**
	gradient	93.43 ± 0.20	**74.21** ± **0.26**	**99.56** ± **0.02**	93.43 ± 0.08
	momentum	93.43 ± 0.08	74.00 ± 0.21	99.52 ± 0.07	93.43 ± 0.08
Sparsity 0.80					
ConvNet	random	**83.84** ± **0.22**	**54.24** ± **0.31**	**99.32** ± **0.06**	91.23 ± 0.23
	gradient	83.47 ± 0.15	53.46 ± 0.06	81.95 ± 35.04	91.23 ± 0.26
	momentum	83.77 ± 0.28	53.62 ± 0.31	82.59 ± 36.89	**91.37** ± **0.14**
MLP	random	**66.73** ± **0.27**	**34.54** ± **0.32**	**97.69** ± **0.10**	**87.88** ± **0.29**
	gradient	64.94 ± 0.46	33.37 ± 0.13	97.59 ± 0.16	87.80 ± 0.39
	momentum	64.72 ± 0.34	33.98 ± 0.07	97.50 ± 0.14	87.79 ± 0.12
ResNet-50	random	92.75 ± 0.17	**71.21** ± **0.31**	99.55 ± 0.04	**93.67** ± **0.23**
	gradient	92.79 ± 0.23	70.79 ± 0.48	99.56 ± 0.02	93.65 ± 0.17
	momentum	**92.86** ± **0.29**	70.75 ± 0.32	**99.60** ± **0.05**	93.60 ± 0.26
ResNet-56	random	92.82 ± 0.18	**70.88** ± **0.44**	**99.57** ± **0.05**	**93.60** ± **0.22**
	gradient	**93.08** ± **0.21**	70.70 ± 0.45	99.59 ± 0.03	93.52 ± 0.21
	momentum	93.04 ± 0.1	70.84 ± 0.48	99.55 ± 0.03	93.56 ± 0.10
VGG-16	random	**93.88** ± **0.09**	74.61 ± 0.19	99.54 ± 0.02	93.44 ± 0.19
	gradient	93.73 ± 0.16	74.64 ± 0.24	**99.57** ± **0.09**	93.44 ± 0.16
	momentum	93.79 ± 0.17	**74.90** ± **0.23**	99.53 ± 0.05	**93.45** ± **0.08**

Table 7: Performance of models tested on various datasets with different sparsity levels, namely 95%, 90%, and 80%.

D RBMs Topological Connectivity

Below some figure that show how the sparse input layers from a big and small RBMs map back to the training examples of the MNIST dataset.

Figure 14: MNIST training examples

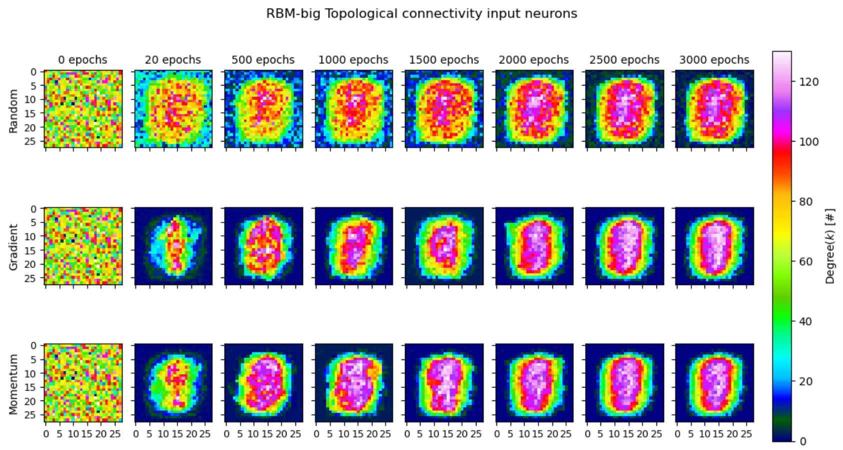


Figure 15: Big RBMs trained on the MNIST dataset, where the rows indicate the regrow strategy used and every subplot (pixels on axis) maps the topological connectivity of the input features to the training examples, averaged over 3 seeds.

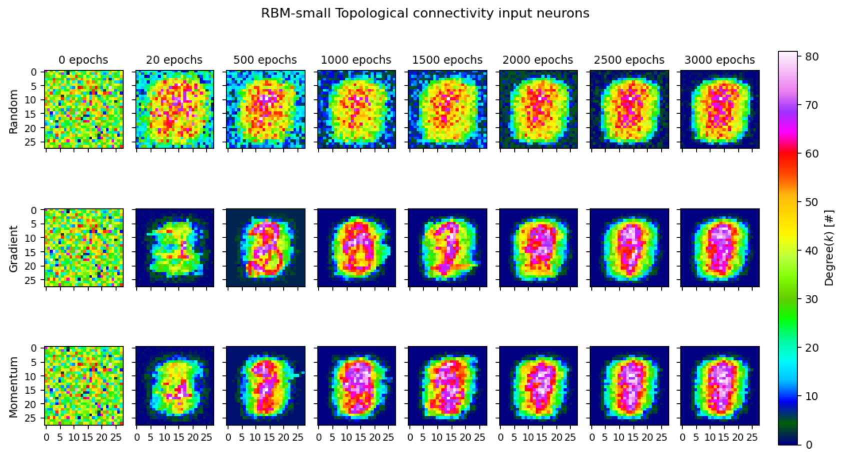


Figure 16: Small RBMs trained on the MNIST dataset, where the rows indicate the regrow strategy used and every subplot (pixels on both axis) maps the topological connectivity of the input features to the training examples, averaged over 3 seeds.

E Mixed regrow strategy

Considering the information previous sections,
we trained models where we switched regrow methods half way through training. The result can be find below in Figure 8. Comparing this with the results obtained from purely training models (Fig. 4) with one regrow method, there is not significant difference.

Models	Methods	CIFAR10	CIFAR100	FashionMNIST
Sparsity 0.95				
ConvNet	random →gradient	76.98 ± 0.28	33.65 ± 18.27	90.34 ± 0.15
	random →momentum	77.29 ± 0.47	33.61 ± 18.23	90.33 ± 0.22
	gradient →random	53.81 ± 13.82	37.81 ± 1.49	65.95 ± 27.15
	gradient →momentum	56.70 ± 15.84	37.66 ± 1.79	66.02 ± 27.22
	momentum →random	73.84 ± 3.19	37.44 ± 1.82	90.01 ± 0.31
	momentum →gradient	74.15 ± 2.99	37.91 ± 0.85	89.97 ± 0.36
MLP	random →gradient	68.28 ± 0.35	36.94 ± 0.38	86.08 ± 0.41
	random →momentum	68.10 ± 0.36	37.25 ± 0.27	86.26 ± 0.41
	gradient →random	65.88 ± 0.18	34.85 ± 0.31	79.63 ± 3.53
	gradient →momentum	66.00 ± 0.33	34.70 ± 0.18	79.63 ± 3.53
	momentum →random	65.94 ± 0.38	35.14 ± 0.21	85.94 ± 0.42
	momentum →gradient	66.18 ± 0.17	35.14 ± 0.28	85.96 ± 0.31
ResNet-56	random →gradient	91.46 ± 0.21	67.29 ± 0.22	93.42 ± 0.07
	random →momentum	91.34 ± 0.29	67.09 ± 0.24	93.50 ± 0.14
	gradient →random	91.02 ± 0.28	65.66 ± 0.74	93.22 ± 0.12
	gradient →momentum	91.20 ± 0.10	65.24 ± 0.21	93.31 ± 0.33
	momentum →random	91.22 ± 0.41	65.39 ± 0.55	93.18 ± 0.09
	momentum →gradient	91.23 ± 0.09	65.58 ± 0.29	93.38 ± 0.29
VGG-16	random →gradient	93.09 ± 0.14	72.33 ± 0.32	93.40 ± 0.07
	random →momentum	93.03 ± 0.18	72.42 ± 0.26	93.49 ± 0.04
	gradient →random	92.99 ± 0.11	72.75 ± 0.15	93.32 ± 0.25
	gradient →momentum	93.22 ± 0.19	72.75 ± 0.15	93.46 ± 0.08
	momentum →random	93.16 ± 0.10	72.81 ± 0.22	93.53 ± 0.19
	momentum →gradient	93.11 ± 0.11	72.70 ± 0.21	93.40 ± 0.18

Table 8: Averaged over 5 runs test results switching regrow strategy at 50% of training (150 epochs) for a subset of the models used in the experiments in Section 5.

F Training samples CIFAR10 dataset

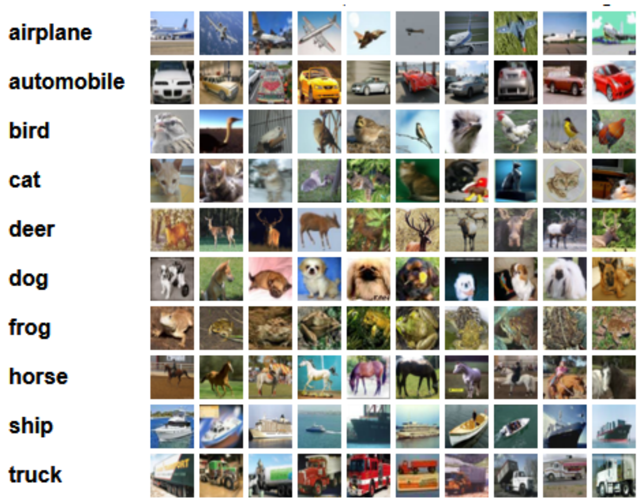


Figure 17: CIFAR10 dataset training samples per class [26].

G Degree distribution input layer

In Figure 19 the distribution of connections regarding the input layer of each sparse MLP is plotted in several subplots. The plots are taken during the training.

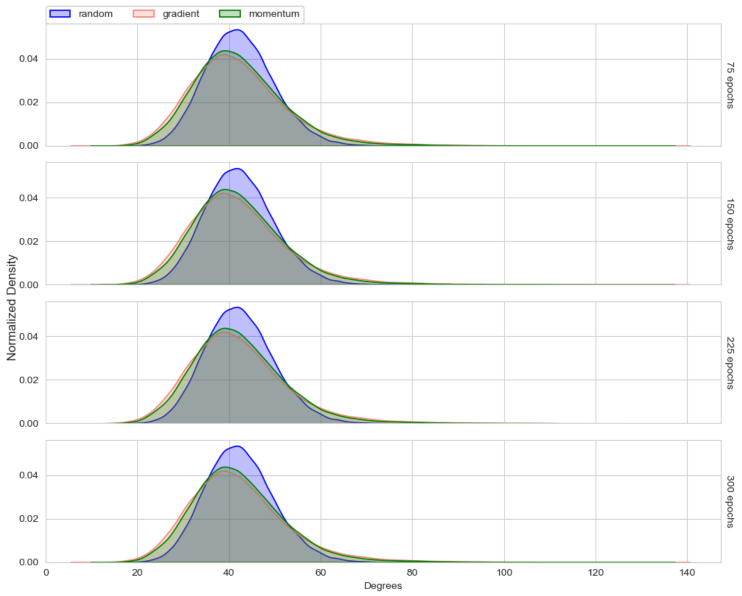


Figure 18: Degree distribution of the input layer over time per different regrow strategy.

H First hidden layer mapped to input layer (topological connectivity)

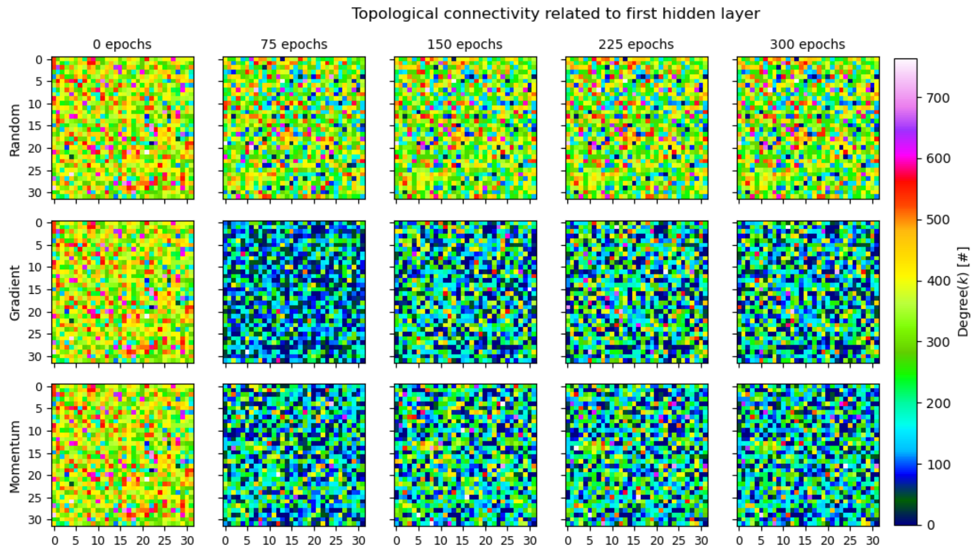


Figure 19: Heatmap visualization of the **first hidden** layer mapped back to the sparse input layer (averaged over 5 seeds). The rows separate regrow methods random, gradient, and momentum, respectively. The columns indicate the state at a specific point during training (epochs). The x- and y-axis of the subplots indicate the pixels.

I Correlation in topology exploration

When evaluating the correlation between two random variables—specifically, the distributions in our context—three prevalent methods are Pearson's, Spearman's rank, and Kendall's correlation coefficient [15, 49, 23]. Each method comes with its distinct set of prerequisites and its own advantages and disadvantages that should guide your choice according to the nature of your data. Pearson's correlation coefficient is ideal for linear and normally distributed changes in data, making it less appropriate for our analysis. In contrast, both Kendall's Tau and Spearman's Rank correlation coefficient are non-parametric, rank-based methods that do not assume normal distribution, aligning them more closely with the requirements of this study.

Kendall's Tau is particularly beneficial when the data involves a significant number of tied ranks, as is the case with neural networks where numerous nodes may evolve to have similar low-value degrees over time. This method's capacity to handle ties—where several data points have the same value—makes it a robust option for datasets where the frequency of ties is expected to increase as the training progresses. Furthermore, Kendall's Tau offers a more nuanced understanding of the ordinal relationship between variables by focusing on the agreement between pairs of data points [17]. Such precision is critical in our study, where the interest lies in the degree to which weaker neurons become progressively weaker.

We used Kendall's Tau to capture the correlation of two random variables during training. In our case the random variables are the degree distributions during training. Figure 20 and Figure 7 resemble similar behaviour in the employed regrow methods. The distributions from the models trained with random are correlating more near the end of training. The models trained with gradient and momentum their distributions, correlate more with their previous structure near the end of training, therefore, implying that these methods find an optimal structure faster - to learn the task given at hand. Also implying topological changes made near the end of training can be considered inefficient, due to the increasing correlation.

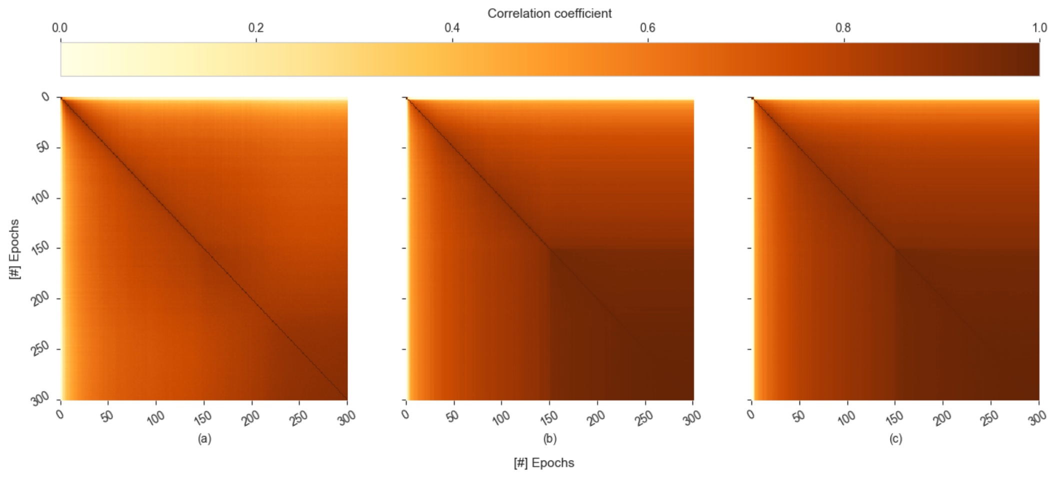


Figure 20: Heatmaps for regrow strategies - (a) random, (b) gradient, and (c) momentum - visualizing the correlation between the degree distributions starting from initialization to post-training. The correlation coefficient ranges from 0 (light) to 1 (dark).