One-Shot Neural Network Pruning via Spectral Graph Sparsification

Steinar Laenen [1]

Abstract

Neural network pruning has gained significant attention for its potential to reduce computational resources required for training and inference. A large body of research has shown that networks can be pruned both after training and at initialisation, while maintaining competitive accuracy compared to dense networks. However, current methods rely on iteratively pruning or repairing the network to avoid over-pruning and layer collapse. Recent work has found that by treating neural networks as a sequence of bipartite graphs, pruning can be studied through the lens of *spectral graph theory*. Therefore, in this work, we propose a novel pruning approach using *spectral sparsification*, which aims to preserve meaningful properties of a dense graph with a sparse subgraph, by preserving the spectrum of the dense graph's adjacency matrix. We empirically validate and investigate our method, and show that one-shot pruning using spectral sparsification preserves performance at higher levels of sparsity compared to its one-shot counterparts. Additionally, we theoretically analyse our method with respect to local and global connectivity.

1. Introduction

Deep neural networks (DNNs) have significantly impacted research and practical applications, but their high parameter count and computational demand present deployment challenges on low-memory devices like mobile phones or smart devices (Huang et al., 2019). Consequently, extensive research is dedicated to improving DNN efficiency, one such direction being neural network pruning (Blalock et al., 2020), which compresses neural networks by removing parameters.

Several works analyse neural network pruning from a probabilistic perspective (Malach et al., 2020), or a gradient-flow perspective (Evci et al., 2022). Recent work (Hoang et al., 2023; Pal et al., 2022) has shown that neural network pruning strategies can be looked at through the lens of *spectral graph theory*, which studies the relationship between the eigenvalues and eigenvectors of a graph's adjacency matrix and the properties of the graph itself. Spectral methods have proven very useful for analysing large graphs that are too complex to be understood by other methods. In particular, the eigenvalues of the graph can be used to determine important properties of the graph, such as its connectivity (Alon, 1986).

Hoang et al. (2023) and Pal et al. (2022) study neural networks by treating them as a sequence of bipartite graphs. By looking at the spectra of the adjacency matrices of these bipartite graphs, they show that 'good' lottery tickets (Frankle & Carbin, 2018) exhibit high graph *expansion*, i.e., these pruned neural networks are sparse yet highly connected. Inspired by their work, we further investigate neural network pruning from a graph connectivity perspective. We start with the observation that many neural network pruning algorithms follow the same high-level algorithm of (i) training a neural network; (ii) producing a score for every edge in the neural network, which quantifies its importance; (iii) retaining $p\%$ of the weights with the highest score. In this work, we focus on the second step of this process. For most scoring methods there is no theoretical guarantee that the network is still connected after greedily selecting the weights with the highest score. Therefore, many approaches take an iterative pruning approach by repeating the three steps above (Frankle & Carbin, 2018), or they repeat steps (ii) and (iii) (de Jorge et al., 2021; Tanaka et al., 2020). In this work we will focus on *one-shot* pruning.

In our method, rather than 'greedily' choosing the weights with the highest score, we aim to generate a pruning mask in one-shot which maintains most of the weights with the highest score, while *simultaneously* maximising the connectivity. In order to do so, we propose to prune neural networks with *spectral sparsification*. Spectral sparsification (Spielman & Teng, 2011) studies the edge sparsification of graphs while maintaining meaningful properties of the

[1]School of Informatics, University of Edinburgh, United Kingdom. Correspondence to: Steinar Laenen <steinar.laenen@ed.ac.uk>.

Proceedings of the 2^{nd} Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023.

spectra of the graph matrices. In particular, by preserving the eigenvalues corresponding to graph matrices, such as the adjacency or Laplacian matrix, one also preserves the overall connectivity patterns across a graph. This observation motivates us to use spectral sparsification for neural network pruning, and we list our contributions below:

Our Contributions

- We prove that spectrally sparsifying the bipartite graphs associated with each layer of the neural network is equivalent to spectrally sparsifying the graph associated on the whole neural network; effectively this means that a *layerwise* pruning approach is provably equivalent to a *global* pruning approach, which is crucial to achieve efficient running times for spectral sparsification on neural networks.
- We therefore propose a layerwise one-shot pruning algorithm for neural networks, either at initialisation or after pre-training, which spectrally sparsifies each layer of the neural network. To the best of our knowledge, this is the first work that uses spectral graph sparsification for neural network pruning.
- We conduct extensive experiments to show that generating masks using spectral sparsification produces better masks than greedy sampling, both in terms of performance and connectivity.

Overview of the Paper In Section 2 we introduce the background on neural network pruning and spectral sparsification. In Section 3 we introduce our method, and in Section 4 we show our experiments. In Section 5 we discuss related work and we conclude the paper in Section 6.

2. Background & Method

This section summarises the background knowledge used in our paper. In Section 2.1 we discuss neural network pruning. In Section 2.2 we introduce basic notations and facts related to spectral graph theory, and in Section 2.3 we formally introduce spectral sparsification of graphs. Finally, in Section 2.4 we explain how we construct graphs from neural network architectures.

2.1. Background on Neural Network Pruning

We will give a brief overview of neural network pruning, and we adopt the notation from (Blalock et al., 2020). We define neural network *architectures* as a family of functions $f(x, :)$, where we include convolution shapes, activation functions, batch normalisation etc. Example architectures are the LeNet (LeCun et al., 1998), VGG (Simonyan & Zisserman, 2014) and ResNet architectures (He et al., 2016).

We let the neural network *model* be an instantiation of a neural network after training, and we denote it by $f(x, W)$, where W are the model parameters. We set the weights to $W^{(0)}$ at initialisation. With $B \in \{0,1\}^{|W|}$ we denote the *mask* of a network, which is used to remove weights from a model. We denote a *pruned* neural network by $f(x, W \odot B)$, where $\odot$ is the element-wise product operator.

In general, most works on neural network pruning adopt the following high-level algorithm:

1. A neural network $f(x, W)$ is trained to convergence.
2. An algorithm produces a score for every edge weight, which quantifies the overall importance of each weight. The weights with the 'worst' score are pruned from the network, which produces a mask $B \in \{0,1\}^{|W|}$ such that the pruned network $f(x, W \odot B)$ is obtained.
3. The pruned network is then fine-tuned by retraining the network only on the unpruned weights, where either the original weights W after training are kept, or they are reinitialised to $W^{(0)}$.

Some works perform this high-level technique once, which is called *one-shot pruning* (Liu et al., 2018). Other methods perform this method iteratively, by sampling a fraction of the weights and retraining over several iterations. These are called *iterative-based pruning methods* (Frankle & Carbin, 2018). Other methods skip the training step and instead prune the network at *initialisation* (Lee et al., 2018). Although in general iterative-based methods perform better than one-shot methods, they are computationally more expensive as the network $f(x, W \odot B)$ has to be trained repeatedly. Therefore, in this paper, we focus on one-shot methods, both at initialization and after convergence.

Several methods have been proposed to 'score' the edges, such as the absolute value of the weights (Han et al., 2015; Frankle & Carbin, 2018), contribution to the gradient, or network activations (Lee et al., 2018). Scores can be given with respect to the whole network, or locally within a layer (Liu et al., 2018). When the neural network is fine-tuned, either the model weights that were used at initialisation are re-used (Frankle & Carbin, 2018), or the network is reinitialised to an earlier state during training (Renda et al., 2019). In this work we reset the model weights to those used at initialisation.

2.2. Preliminaries on Spectral Graph Theory

We always assume that $G = (V, E, w)$ is an undirected graph with $|V| = n$ vertices, $|E| = m$ edges, and weight function $w : V \times V \to \mathbb{R}_{\geqslant 0}$. For any edge $e = \{u, v\} \in E$, we write w_e or w_{uv} to express the weight of e. For a vertex $u \in V$, we denote its *degree* by $d_u \triangleq \sum_{v \in V} w_{uv}$. For

a graph $G = (V, E, w)$, let $\mathbf{D} \in \mathbb{R}^{n\times n}$ be the diagonal matrix defined by $\mathbf{D}_{uu} = d_u$ for all $u \in V$. We denote by $\mathbf{A} \in \mathbb{R}^{n\times n}$ the *adjacency matrix* of G, where $\mathbf{A}_{uv} = w_{uv}$ for all $u, v \in V$. We then define the *Laplacian* matrix as $L_G = D - A$. The *normalised Laplacian matrix* of G is defined as $\mathcal{L} \triangleq \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{I}$ is the $n \times n$ identity matrix. For any input graph $G = (V, E, w)$ and any $S \subset V$, let the conductance of S in G be

$$\Phi_G(S) \triangleq \frac{w(S, V \setminus S)}{\mathrm{vol}(S)},$$

where $w(S, T) \triangleq \sum_{e\in E(S,T)} w_e$ is the cut value of (S, T) and $\mathrm{vol}(S) \triangleq \sum_{u\in S} d_u$ is the volume of the set S. We define the *conductance* of G by

$$\Phi_G \triangleq \min_{\substack{S\subset V \\ \mathrm{vol}(S)\leqslant \mathrm{vol}(V)/2}} \Phi_G(S).$$

Many well-known properties of graphs and their random walks are understood through the spectra of Laplacians. See e.g. Chung (1997) for details. A celebrated example of this is the Cheeger inequality:

Lemma 2.1 (Cheeger Inequality, (Alon, 1986))**.** *It holds for any graph G that*

$$\frac{\lambda_2}{2} \leqslant \Phi_G \leqslant \sqrt{2\lambda_2}.$$

Here, λ_2 is the second eigenvalue of the normalised Laplacian $\mathcal{L}_G$. At a high-level, the Cheeger inequality states that if λ_2 is small, then the graph expansion, i.e. its connectivity, is also low. On the other hand, if λ_2 is large, then its connectivity is high. In our setting, it will be useful to consider λ_2 as a measure of the connectivity of a graph/neural network.

2.3. Spectral Sparsification

Graph sparsification studies an effective representation of an undirected graph G by a sparse subgraph H of G, such that meaningful properties of G are preserved in H. Since most graph algorithms run faster on sparser graphs, this problem has received great interest in recent years. Our focus will be on *spectral sparsification*. In this setting, we are interested in preserving the spectral properties with respect to the Laplacian $\mathcal{L}_G$. We formally define spectral sparsification below.

Definition 2.2 (Spectral Sparsification)**.** *Given a weighted, undirected graph $G = (V, E, w)$, and an error parameter $\varepsilon \in (0, 1)$, we say that a weighted subgraph $H = (V, \widetilde{E}, \widetilde{w})$, $\widetilde{E} \subset E$, is an ε-sparsifier of G if for all $x \in \mathbb{R}^n$ we have that*

$$(1 - \varepsilon) \cdot x^\intercal L_G x \leqslant x^\intercal L_H x \leqslant (1 + \varepsilon) \cdot x^\intercal L_G x,$$

where L_G and L_H are the Laplacian matrices of G and H respectively.

A well-known fact in spectral graph theory is that spectral sparsifiers preserve eigenvalues related to the Laplacian matrix. One can readily show by the Courant-Fisher characterisation of eigenvalues, that

$$(1 - \varepsilon) \cdot \lambda_k(L_G) \leqslant \lambda_k(L_H) \leqslant (1 + \varepsilon) \cdot \lambda_k(L_G),$$

for all $1 \leqslant k \leqslant n$, where $\lambda_k \in \mathbb{R}$ is the k-th largest eigenvalue of either L_H or L_G. Therefore, by approximately preserving the eigenvalues of the normalised Laplacian, we also approximately preserve the graph expansion, i.e., its connectivity. There are several works that present algorithms for constructing ε-spectral sparsifiers (Batson et al., 2012; Lee & Sun, 2018; Spielman & Srivastava, 2011; Spielman & Teng, 2011). In this paper, we will focus on spectral sparsification by effective resistances (Spielman & Srivastava, 2011).

Spectral sparsification by effective resistances is one of the simplest sparsification algorithms. We formally describe it in Algorithm 1. At a high level, sparsification by effective resistances works by treating a graph G as an electrical network. We then define the effective resistance over an edge $e = \{u, v\} \in E(G)$ as

$$\mathrm{Reff}(u, v) \triangleq (\delta_u - \delta_v)^\intercal L_G^\dagger (\delta_u - \delta_v),$$

where $\delta_u \in \{0, 1\}^n$ is the indicator vertex of u, and $L_G^\dagger$ is the pseudo-inverse of L_G. Effective resistances are then a measure of how difficult it is for electrical current to flow between two nodes in a graph. To sparsify G, we visit every edge $e \in E(G)$ and include it in the sparsified graph G with probability proportional to $\ell(e) \triangleq \mathrm{Reff}(e) \cdot w_e$.

Algorithm 1 Sparsification by Effective Resistances (SpecSpar)

Input: a weighted graph $G = (V, E, w)$ & sparsification parameter $\varepsilon \in \mathbb{R}^+$.
$n = |V|$
Output: A sparsified graph $H = (V, \widetilde{E}, \widetilde{w})$
for $e \in E$ **do**
  let $\ell_e = w(e) \cdot \mathrm{Reff}(e)$
  let $p_e = \min\left(1, 5 \cdot \log n \cdot \ell_e/\varepsilon^2\right)$
end for
$H = (V, \emptyset)$
for $e \in E$ **do**
  with probability p_e add edge e into H with weight w_e/p_e
end for
Return: H

The performance of their algorithm is summarised in the following Theorem:

Theorem 2.3 (Spielman & Srivastava (2011))**.** *Let G be any undirected graph with n vertices. For any $\varepsilon \in (0, 1)$,*

Algorithm 1 produces a $(1+\varepsilon)$-spectral sparsifier of G with $O(n \log n/\varepsilon^2)$ edges.

We remark that other algorithms exist that run faster, and produce sparsifiers with fewer edges (Batson et al., 2012; Lee & Sun, 2018). However, these algorithms use complicated subroutines which are difficult to implement (e.g., semidefinite programming), and therefore we choose to work with spectral sparsification by effective resistances.

To explain the intuition behind Algorithm 1, if both $\mathrm{Reff}(e)$ and w_e are high/low, then we include the edge with high/low probability. Now consider the case where $w(e)$ is high but $\mathrm{Reff}(e)$ is low. This means that the edge e does not contribute much to the overall connectivity of the network, and therefore is sampled with lower probability *even if* $w(e)$ *is high*. On the other hand, in the case where $w(e)$ is low but $\mathrm{Reff}(e)$ is high, it means that e is important to the network connectivity, even if $w(e)$ is low. We illustrate in Figure 1 a small example of a neural network that is being pruned by either choosing edges with the highest weight, or being sparsified by Algorithm 1.

2.4. Characterising Neural Networks as Graphs

Before we introduce our method, we first explain how to construct bipartite graphs from neural network layers, which we adopt from (Pal et al., 2022). For a *linear/fully-connected layer* $W_i \in \mathbb{R}^{\kappa_i \times \kappa_{i+1}}$, we simply construct the bipartite graph $G = (L \cup R, E, w)$ with $|L| = \kappa_i$, $|R| = \kappa_{i+1}$, and the weight $w(e)$ for $e = \{u, v\} \in E$ is determined by the corresponding entry in the score matrix $S_i(u, v)$.

For a *convolutional layer* with parameters kernel height K_h, kernel width K_w, output channels C_{out} and input channels C_{in}, we construct the corresponding bipartite graph $G = (L \cup R, E, w)$ by unfolding the input parameters to get $W_i \in \mathbb{R}^{|L| \times |R|}$ such that $|L| = K_h \times K_w \times C_{\text{in}}$ and $|R| = C_{\text{out}}$, and the weight $w(e)$ for $e = \{u, v\} \in E$ is determined by the corresponding entry in the score matrix $S_i(u, v)$.

Given a neural network $f(x, W)$ with ℓ layers and scores $S = \{S_1, \dots S_\ell\}$, we then construct the bipartite graph for each layer, and get a set of ℓ graphs $\mathcal{G} = \{G_1, \dots G_\ell\}$.

3. Algorithm and Analysis

Consider a neural network $f(x, W)$ with ℓ layers $\mathcal{G} = \{G_1, \dots G_\ell\}$, where for simplicity we assume that each layer is fully-connected. Given Algorithm 1, a natural first approach to prune the full network $f(x, W)$ would be to construct the full ℓ-partite graph $G_{\text{full}} = \bigcup_{i=1}^{\ell} G_i$, and run Algorithm 1 directly on G_{full}. Unfortunately, this is computationally too expensive. The computation of the effective resistances $\mathrm{Reff}(e)$ in line 1 of Algorithm 1 increases at least polynomially in the number of nodes. This is due to the computation of the pseudoinverse $L^{\dagger}_{G_{\max}}$, which takes $O(n^\omega)$ time, where $n = |V(G_{\text{full}})|$ and ω is the matrix multiplication constant (Alman & Williams, 2021). As modern neural network architectures (e.g., VGG, ResNet) are very large, containing $10,000$s of nodes/neurons, this means that Algorithm 1 is too slow for the purpose of neural network pruning, which we empirically verify in Section 4.

3.1. Our Proposed Algorithm

Therefore, to circumvent this issue, we propose an algorithm (Sparsify) to prune neural networks in a *layerwise* fashion. At a high-level, Sparsify first computes the scores $\{S_1, \dots, S_\ell\}$ using the score function θ for all the edges in every layer. The score functions that we will consider are (i) magnitude (Frankle & Carbin, 2018), (ii) SNIP (Lee et al., 2018), and (iii) SynFlow (Tanaka et al., 2020). Next, Sparsify loops through every layer of the network S_i, and constructs the graph G_i based on the layer structure and scores. We perform spectral sparsification on G_i to obtain H_i, which is then converted into a mask B. Finally, we return the pruned network $f(x, W \odot B)$. A formal description of Sparsify can be found in Algorithm 2.

Algorithm 2 Pruning with Spectral Sparsification (Sparsify)

Input: a NN $f(x, W)$, sparsification parameter $\varepsilon \in \mathbb{R}^+$, score function θ.
Output: A pruned NN $f(x, W \odot B)$
$\ell \leftarrow |W|$
$\{S_1, \dots S_\ell\} \leftarrow \theta(W)$ {Compute edge scores}
$B \leftarrow \emptyset$ {Initialize mask}
for $i \in \{1, \dots, \ell\}$ **do**
  Construct graph G_i from score S_i
  $H_i \leftarrow \mathsf{SpecSpar}(G_i, \varepsilon)$ {Spectrally sparsify G_i}
  Create mask B_i from graph H_i
  $B \leftarrow B \cup B_i$
end for
Return: $f(x, W \odot B)$

We first highlight that the running time for Algorithm 2 is $O(\ell \cdot n_{\max}^{\omega})$, where $n_{\max} \triangleq \max_i |V(G_i)|$ is the largest number of nodes in a single layer, and ℓ is the number of layers. This is significantly faster than Algorithm 1, as modern neural network architectures typically contain many layers, with a limited number of neurons in each layer, e.g., ResNets (He et al., 2016).

3.2. Theoretical Analysis

As mentioned at the beginning of this section, ideally, we would sparsify the graph G constructed from the *whole* network at once, in order to preserve the connectivity patterns of the entire neural network. It has been shown that global

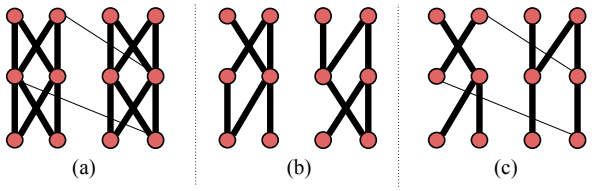

Figure 1. Examples of small neural networks, where the density of an edge corresponds to the score it was given, e.g. magnitude. (a) is the network that is being pruned. (b) shows a pruned network where 12 weights with the highest score were chosen, disconnecting the network. (c) shows a pruned network where edges were chosen using spectral sparsification. The two edges with low edge weight crossing the two highly connected parts of the network have high effective resistance in this network.

pruning methods in general perform better performance-wise than layerwise pruning methods (Blalock et al., 2020).

However, we show that our layerwise pruning approach is equivalent to a global one. Due to the nice properties of spectral sparsification, we prove that if we compute an ε-sparsifier H_i for every layer $G_i \in \mathcal{G}$, then the resulting layerwise sparsified graph $H = \bigcup_i^\ell H_i$ over the entire network is an ε-sparsifier of G. Crucially, this means that by *locally* preserving the connectivity structure, we also preserve the connectivity structure of the whole graph. We can therefore sparsify each G_i individually, which is computationally more efficient than pruning G in one go, and it motivates our design of Algorithm 2. We prove the following Lemma.

Lemma 3.1. *Let $G = (V, E, w)$ be the graph such that $G = \bigcup_i^\ell G_i$ represents a neural network $f(x, W)$, and for every $1 \leqslant i \leqslant \ell$ let H_i be an ε-sparsifier of G_i. Then it holds that $H = \bigcup_i^\ell H_i$ is an ε-sparsifier of G.*

Proof. Notice that we can write the Laplacian matrix of H by summing together the Laplacian matrices of every H_i for $1 \leqslant i \leqslant \ell$, i.e.,

$$L_H = \widetilde{L}_{H_1} + \ldots + \widetilde{L}_{H_\ell},$$

where each $\widetilde{L}_{H_i} \in \mathbb{R}^{n \times n}$ is the Laplacian L_{H_i} of H_i, with added zero columns & vectors for the vertices $u \in G$ such that $u \notin H_i$. Let $x \in \mathbb{R}^n$ be a vector. Then we compute

$$\begin{aligned} x^\intercal L_H x &= x^\intercal \left(\widetilde{L}_{H_1} + \ldots + \widetilde{L}_{H_\ell} \right) x \\ &= x^\intercal \widetilde{L}_{H_1} x + \ldots + x^\intercal \widetilde{L}_{H_\ell} x \\ &= x_{H_1}^\intercal \widetilde{L}_{H_1} x_{H_1} + \ldots + x_{H_\ell}^\intercal \widetilde{L}_{H_\ell} x_{H_\ell}, \end{aligned} \tag{1}$$

where x_{H_i} is the vector x restricted to the vertices $V(H_i)$, i.e. we set $x_u = 0$ for $u \notin V(H_i)$. Because L_{H_i} is an ε-sparsifier of G_i it holds that $x_{H_i}^\intercal \widetilde{L}_{H_i} x_{H_i} \leqslant (1+\varepsilon) \cdot x_{G_i}^\intercal \widetilde{L}_{G_i} x_{G_i}$. From (1) we therefore have that

$$\begin{aligned} x^\intercal L_H x &\leqslant (1+\varepsilon) \cdot \left(x_{G_1}^\intercal \widetilde{L}_{G_1} x_{G_1} + \ldots + x_{G_\ell}^\intercal \widetilde{L}_{G_\ell} x_{G_\ell} \right) \\ &= (1+\varepsilon) \cdot x^\intercal \left(\widetilde{L}_{G_1} + \ldots + \widetilde{L}_{G_\ell} \right) x \\ &= (1+\varepsilon) \cdot x^\intercal L_G x. \end{aligned}$$

Similarly, we can show that $x^\intercal L_H x \geqslant (1-\varepsilon) \cdot x^\intercal L_G x$. This completes the proof that H is an ε-sparsifier of G. □

Time Complexity Let $n_{\max} = V(G_{\max})$ be the number of vertices in the largest graph $G_{\max} \in \mathcal{G}$ corresponding to each layer in the neural network. The running time of Algorithm 2 is dominated by the computation of the effective resistances $\mathrm{Reff}(e)$ in line 1 of Algorithm 1, due to the computation of the pseudoinverse $L_{G_{\max}}^\dagger$. Computing the pseudoinverse takes $O(n^\omega)$ time, where ω is the matrix multiplication constant (Alman & Williams, 2021). One can also get approximate values for the effective resistances in faster running time (Kyng & Sachdeva, 2016), however, these algorithms are quite involved to implement compared to directly using solvers from Python libraries.

Given this, by sequentially sparsifying the layers of the neural network as done in Algorithm 2, we can upper bound the running time as $O(\ell \cdot n_{\max}^\omega)$, where ℓ is the number of layers/bipartite graphs in $\mathcal{G}$. This is how we implemented the algorithm, however, given enough memory, one could speed up the algorithm by sparsifying each layer in parallel, such that the factor of ℓ drops in the running time.

3.3. Comparison to Previous Work

Compared to previous work on neural network pruning/LTH using spectral graph theoretic techniques, we first highlight that Hoang et al. (2023) introduce a connectivity measure based on the third largest eigenvalue, and do not provide any algorithm for pruning at initialisation. On the other hand, Pal et al. (2022) propose an adjustment to the iterative pruning algorithm to ensure that each layer in the pruned network still has high connectivity. They introduce a termination criterion: pruning for a layer is halted once the spectral gap between the first and second eigenvalue of the adjacency matrix for G_i drops below a predetermined value. However, this limits how much a network can be pruned, because stopping the pruning of specific layers prevents the algorithm from reaching arbitrary sparsity. Furthermore, the spectral gap might decrease drastically due to the naive greedy method of retaining the top $p\%$ highest weights in a layer. Lastly, their method does not work in the one-shot setting; if a network is only pruned once, it is not possible to check which layers have lost their Ramanujan property. Our proposed algorithm improves upon theirs in all these aspects: it is *one-shot*, prunes every layer up to *arbitrary sparsity*, and *maximises connectivity*.

Finally, we remark that Hoang et al. (2023) and Pal et al. (2022) only analysed the spectral properties of each bipartite layer separately. Our analysis is the first to show that layerwise pruning is equivalent to global pruning.

4. Experiments

In Section 4.1 we verify that Algorithm 2 (Sparsify) is significantly faster than Algorithm 1, while achieving similar pruning results. In Section 4.2 we empirically evaluate Sparsify, and compare it on several computer vision classification benchmarks, across architectures and datasets. Section 4.3 investigates the sparsity of every layer as measured by the second eigenvalue of the normalised Laplacian matrix.

We will follow standard experimental procedures in the literature (Blalock et al., 2020). The image datasets that will be used for model evaluation are CIFAR-10/100 (Krizhevsky & Hinton, 2009), and *tiny*ImageNet (Le & Yang, 2015). The neural network architectures that will be tested are variants of ResNet (He et al., 2016) and VGG (Simonyan & Zisserman, 2014). In particular, we consider the following architecture/dataset combinations: VGG-11 (on CIFAR-10), VGG-16 (on CIFAR-100), ResNet-20 (on CIFAR-10), and ResNet-18 (on *tiny*ImageNet). A detailed description of benchmarks, architectures and choice of hyperparameters is deferred to the Appendix.

4.1. Layerwise vs Global Spectral Sparsification

To empirically justify the benefit of doing layerwise pruning using spectral sparsification, we look at the pruning at initialisation setting for CIFAR100, trained using VGG16 where we use the edge magnitudes as a score function. We then generate a 1% density pruning mask using both Algorithm 1 on the ℓ-partite graph corresponding to the entire neural network with ℓ layers, and Algorithm 2 which prunes the neural network in a layerwise fashion. We perform the same comparison on a ResNet20 trained on CIFAR-10, but where we sample 3% of the edges. Results can be found in Table 1. For both comparisons, the layerwise and global pruning approaches achieve similar accuracies. However, the layerwise pruning approaches are significantly faster than the global pruning approaches.

Pruning Method	**Pruning Time**	**Accuracy**
VGG-16 CIFAR-100		
Layerwise	321 seconds	59.81 ± 0.21
Global	4321 seconds	59.62 ± 0.32
ResNet20 CIFAR-10		
Layerwise	61 seconds	77.21 ± 0.21
Global	1620 seconds	77.49 ± 0.39

Table 1. Running time and accuracy comparisons between Algorithm 1 (global) and Algorithm 2 (layerwise) in the pruning at initialisation setting for VGG-16 trained on CIFAR-100, pruned up to 1%, and Resnet20 trained on CIFAR-10 pruned up to 3%.

4.2. Pruning at Initialisation

We now evaluate our algorithm in the pruning at initialisation setting. We compare our algorithm against a modification of Algorithm 2. Instead of spectral sparsification for every layer (line 8 of Algorithm 2), we instead keep the edges that correspond to the $p\%$ highest scores. This is the same method that (Pal et al., 2022) compare against. We call this adjustment Greedy. We report the results for all the aforementioned architecture/dataset combinations, and they are reported in Figure 2.

We see that Sparsify in general performs better than Greedy across all scoring methods. For magnitude based pruning on all datasets, the greedy approach performs similarly at high densities, but worse at high levels of sparsity. In general, for SNIP100, layer collapse occurs, and it does not perform well. On all the *tiny*ImageNet results we see that Greedy performs better than Sparsify at extreme sparsities; this might be because layerwise collapse might still occur at high very high sparsity levels, where the spectral sparsification guarantees of Theorem 2.3 no longer apply.

We also observe that the scores from the SynFlow method generally perform best across dataset/architecture combinations. A possible explanation might be that the combination of our proposed algorithm and the SynFlow scores is good at preserving connectivity in the network, because the scores generated by SynFlow are computed to maximise synaptic connectivity.

4.3. Layer connectivity

Next, we investigate whether spectral sparsification indeed preserves layerwise connectivity. Instead of evaluating models with pruning at initialisation, we look at one-shot pruning after pretraining. Even though sparse lottery tickets should exist at initialisation, after convergence the structure in the network might exhibit more higher-order patterns that spectral sparsification might preserve. In particular, we compare pruned models trained on CIFAR-10 with the VGG11 architecture. Since SynFlow and SNIP100 are specifically designed for pruning at initialisation, we only investigate scores based on the weight magnitudes.

We compare our algorithm using weight magnitudes as the score function (Sparsify) against the greedy algorithm using magnitudes as the score function (MagPrune). Results are reported in Figure 3. In the left figure, we report the

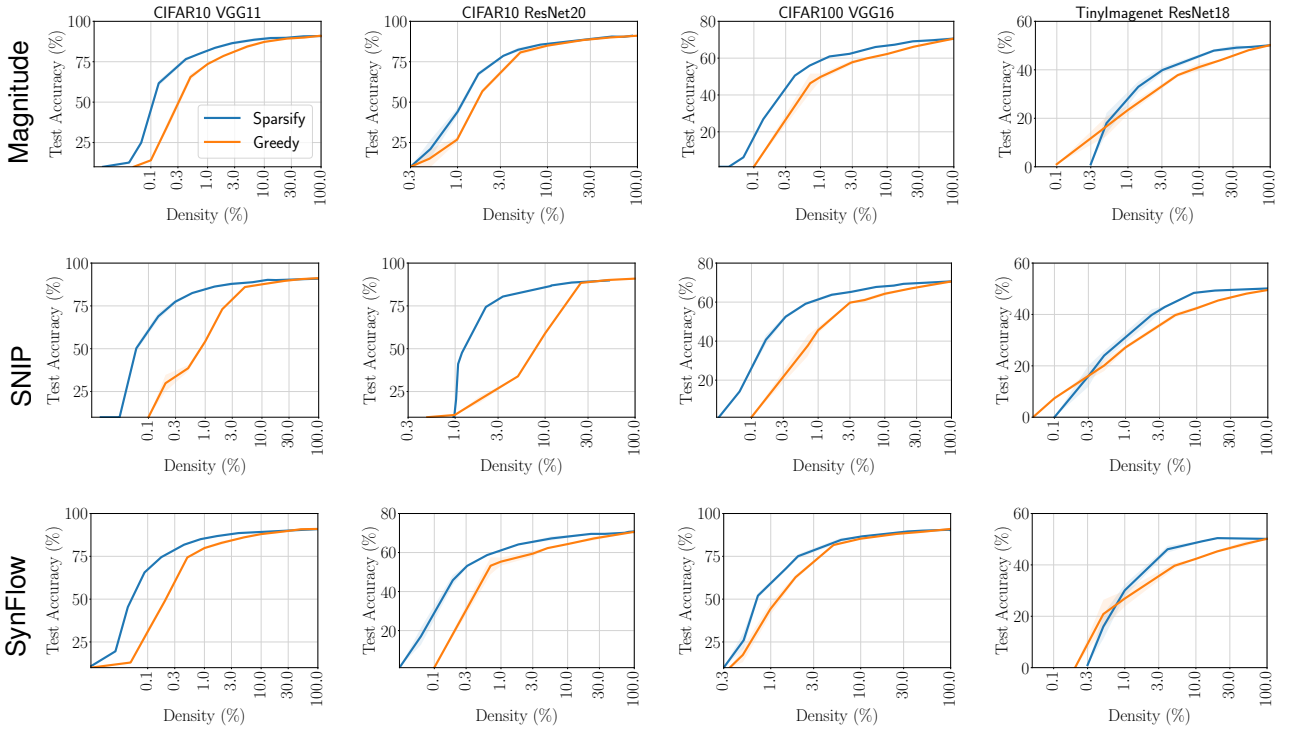

Figure 2. Results for pruning at initialisation. Every row corresponds to a scoring method: Magnitude, SNIP or SynFlow. Every column corresponds to an architecture + dataset combination: CIFAR10 + VGG11, CIFAR10 + ResNet20, CIFAR100 + VGG16, ResNet18 + TinyImagenet. Our method Sparsify corresponds to the blue lines, and the Greedy edge selection method corresponds to the orange lines. For more details see Section 4.2

accuracy at different densities. In the right figure, we plot the value of λ_2, the second eigenvalue of the normalised Laplacian. Note that at higher sparsity levels the graphs become disconnected, and therefore we report λ_2 for the largest connected component in the layer. We see that for both methods connectivity decreases as the network becomes sparser, which is expected. However, we can observe that the greedy pruning approach 'collapses' layer 10, leading to collapse in the accuracy. On the other hand, this does not happen for our method, and hence we manage to retain a higher level of performance at higher sparsity levels. We emphasise that our layerwise pruning method outperforms/matches the performance of the global pruning approach, and outperforms them at high sparsity levels.

We refer to Appendix B for a visualisation of all the layers for Figure 3.

5. Related Work

Neural network pruning aims to reduce unnecessary parameters/components in neural networks, thereby lessening the memory demand during training or deployment (Frankle & Carbin, 2018; Han et al., 2015; LeCun et al., 1989; Molchanov et al., 2017; Mozer & Smolensky, 1989). As neural networks grow in size, research focuses on finding efficient networks either before training (pruning at initialisation) or one-shot post-training. SNIP was among the first to prune at initialisation, introducing a connection sensitivity-based saliency criterion to identify and remove non-essential parameters before training (Lee et al., 2018). Several works (Sreenivasan et al., 2022; Tanaka et al., 2020; Wang et al., 2020) introduced different pruning criteria/methods for identifying high-importance weights, with Wang et al. (2020) maximising gradient flow and Tanaka et al. (2020) introducing a synaptic flow-based saliency criteria. Alizadeh et al. (2022) proposed meta-gradients for initialisation pruning. An extensive comparison by Frankle et al. (2021) revealed these methods generally underperform those training post-pruning. Meanwhile, Miao et al. (2021) employed stochastic Frank-Wolfe optimisation for one-shot pruning without retraining. Chen et al. (2021) developed a DNN compression framework by partitioning DNN parameters into zero-invariant groups and promoting zero groups via structured-sparsity optimisation.

Graph theory techniques are increasingly applied in analysing and constructing neural networks. You et al. (2020) analysed neural networks' predictive properties using relational subgraphs, while Mocanu et al. (2018) and Evci

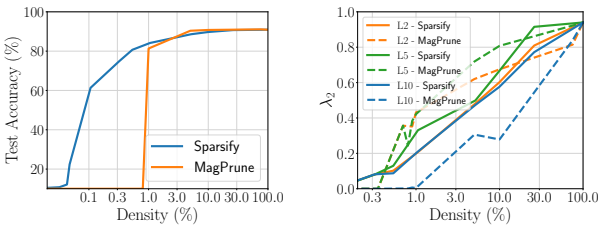

Figure 3. In the left figure, we plot the accuracy of VGG-11 trained on CIFAR-10 at different densities. We compare single-shot magnitude pruning (MagPrune) to Sparsify, using magnitudes as the score function. In the right figure we plot the value of λ_2, for different layers in each of the methods. Here, colors represent layers and dotted lines correspond to MagPrune vs solid lines which correspond to Sparsify.

et al. (2020) utilized Paul Erdős-Réyni graphs to adjust score distribution between layers. Vooturi et al. (2021) introduced Ramanujan bipartite graph products for layer construction to optimise density and connectivity while improving running time. Stewart et al. (2023) recently improved on this work by proposing stronger models that generate better expanders to perform data-free pruning at initialisation. The relationship between network graph topology and skip connections was studied by Bhardwaj et al. (2021).

Our work is mainly influenced by the recent studies by Pal et al. (2022) and Hoang et al. (2023), who explored both the Lottery Ticket Hypothesis (Frankle & Carbin, 2018) and pruning at initialisation with spectral graph theory. They used Ramanujan graphs and corresponding eigenvalue bounds to propose criteria assessing the 'connectivity' of the pruned subnetwork. Their findings highlighted that successful lottery tickets and high-performing subnetworks exhibit Ramanujan graph properties. Our work differs primarily by: (i) relaxing the spectral constraints and just considering λ_2 of the Laplacian as a graph connectivity measure instead of strict Ramanujan bounds; (ii) proposing an algorithmic framework using this relaxed constraint to locate sparse networks inside densely initialised ones, maintaining high accuracy at high sparsity levels. In their work, Pal et al. (2022) only propose a stopping criteria for pruning, and Hoang et al. (2023) do not have any algorithmic contribution.

6. Conclusion

In this work, we have shown that spectral sparsification is a suitable method for choosing masks in one-shot neural network pruning, confirming prior work which found a relationship between graph spectra and pruned neural network performance. In particular, we have shown empirically that in the one-shot setting our proposed algorithm outperforms the widely-used method of choosing the $p\%$ highest weights in the network. Moreover, and perhaps as expected, we found that our method preserves the spectral properties of the Laplacian matrices of neural network layers. Beyond the context of this work, we believe that our new mask selection approach and its insights might have impact in the design of future neural network pruning algorithms.

References

Alizadeh, M., Tailor, S. A., Zintgraf, L. M., van Amersfoort, J., Farquhar, S., Lane, N. D., and Gal, Y. Prospect pruning: Finding trainable weights at initialization using meta-gradients. In *International Conference on Learning Representations (ICLR'22)*, 2022.

Alman, J. and Williams, V. V. A refined laser method and faster matrix multiplication. In *32st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*, 2021.

Alon, N. Eigenvalues and expanders. *Combinatorica*, 6(2): 83–96, 1986.

Batson, J., Spielman, D. A., and Srivastava, N. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41 (6):1704–1721, 2012.

Bhardwaj, K., Li, G., and Marculescu, R. How does topology influence gradient propagation and model performance of deep networks with densenet-type skip connections? In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 13498–13507, 2021.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Chen, T., Ji, B., Ding, T., Fang, B., Wang, G., Zhu, Z., Liang, L., Shi, Y., Yi, S., and Tu, X. Only train once: A one-shot neural network training and pruning framework. In *Advances in Neural Information Processing Systems 34 (NeurIPS'21)*, 2021.

Chung, F. R. K. *Spectral Graph Theory*. American Mathematical Society, 1997.

de Jorge, P., Sanyal, A., Behl, H. S., Torr, P. H., Rogez, G., and Dokania, P. K. Progressive skeletonization: Trimming more fat from a network at initialization. In *International Conference on Learning Representations (ICLR'21)*, 2021.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *37th International Conference on Machine Learning (ICML'20)*, 2020.

Evci, U., Ioannou, Y., Keskin, C., and Dauphin, Y. Gradient flow in sparse neural networks and how lottery tickets

win. In *36th AAAI Conference on Artificial Intelligence (AAAI'22)*, 2022.

Frankle, J. Openlth: A framework for lottery tickets and beyond. https://github.com/facebookresearch/open_lth#openlth-a-framework-for-lottery-tickets-and-beyond, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR'18)*, 2018.

Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations (ICLR'21)*, 2021.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hoang, D. N., Liu, S., Marculescu, R., and Wang, Z. Revisiting pruning at initialization through the lens of ramanujan graph. In *International Conference on Learning Representations (ICLR'23)*, 2023.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems 32 (NeurIPS'19)*, 2019.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Kyng, R. and Sachdeva, S. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS'16)*, pp. 573–582. IEEE, 2016.

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In *Advances in Neural Information Processing Systems 2 (NeurIPS'89)*, 1989.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, N., Ajanthan, T., and Torr, P. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR'18)*, 2018.

Lee, Y. T. and Sun, H. Constructing linear-sized spectral sparsification in almost-linear time. *SIAM Journal on Computing*, 47(6):2315–2336, 2018.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR'18)*, 2018.

Malach, E., Yehudai, G., Shalev-Schwartz, S., and Shamir, O. Proving the lottery ticket hypothesis: Pruning is all you need. In *37th International Conference on Machine Learning (ICML'20)*, 2020.

Miao, L., Luo, X., Chen, T., Chen, W., Liu, D., and Wang, Z. Learning pruning-friendly networks via frank-wolfe: One-shot, any-sparsity, and no retraining. In *International Conference on Learning Representations (ICLR'21)*, 2021.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9 (1):2383, 2018.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR'17)*, 2017.

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

Pal, B., Biswas, A., Kolay, S., Mitra, P., and Basu, B. A study on the ramanujan graph property of winning lottery tickets. In *39th International Conference on Machine Learning (ICML'22)*, 2022.

Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations (ICLR'19)*, 2019.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Spielman, D. A. and Srivastava, N. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6): 1913–1926, 2011.

Spielman, D. A. and Teng, S.-H. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

Sreenivasan, K., Sohn, J.-y., Yang, L., Grinde, M., Nagle, A., Wang, H., Lee, K., and Papailiopoulos, D. Rare gems: Finding lottery tickets at initialization. In *Advances in Neural Information Processing Systems 36 (NeurIPS'22)*, 2022.

Stewart, J., Michieli, U., and Ozay, M. Data-free model pruning at initialization via expanders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4518–4523, 2023.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems 33 (NeurIPS'20)*, 2020.

Vooturi, D. T., Varma, G., and Kothapalli, K. Ramanujan bipartite graph products for efficient block sparse neural networks. *Concurrency and Computation: Practice and Experience*, pp. 6363, 2021.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations (ICLR'20)*, 2020.

You, J., Leskovec, J., He, K., and Xie, S. Graph structure of neural networks. In *37th International Conference on Machine Learning (ICML'20)*, pp. 10881–10891, 2020.

A. Omitted Experiment Details

Training Details We adopt the hyperparameters found in Frankle et al. (2021). For CIFAR-10 and CIFAR-100, we train all models for 160 epochs, with initial learning rate 0.1; we decrease the learning rate by a factor of 10 at epoch 80 and 120. Weight decay was set to 1×10^{-4} and the batch size was 256. We augment training data by normalizing per channel, randomly flipping horizontally, and randomly shifting by up to four pixels in any direction. For *tiny*ImageNet we train for 200 epochs, with initial learning rate 0.2, and we decrease the learning rate by a factor of 10 at epoch 100 and 150. We train with a batch size of 256, and we augment it by normalizing per channel, selecting a patch with random aspect ratio between 0.8 and 1.25, and random scale between 0.1 and 1, cropping to 64×64 and randomly flipping horizontally.

All experiments were ran on a single 8GB NVIDIA GeForce RTX 2080. We repeat experiments three times with different seeds and report the average result over three runs and the confidence interval. We used the OpenLTH package for implementing methods and running experiments (Frankle, 2020).

Implementation details In order to prune networks to arbitrary sparsity, we slightly adjust the sampling probability in line 1 of Algorithm 1 to be $p_e = \min(1, 5 \cdot \log n \cdot \ell_e \cdot \varepsilon)$. This way we can keep decreasing ε, rather than having $\varepsilon = 1$ be a lower bound for our sampling probability. Even though this effects the theoretical guarantees of Algorithm 1, we still find that our algorithm performs well in practice. Moreover, if a specific target sparsity q is stated for Algorithm 1, we choose epsilon such that the number of edges sampled is slightly below q, and then we randomly choose a number of edges such that the sparsity of the network is exactly q. For SynFlow we set the number of pruning iterations to 100. For SNIP we set the number sampled used to 100. We refer to Tanaka et al. (2020) and Lee et al. (2018) for implementation details.

B. Additional Results Layerwise Connectivity

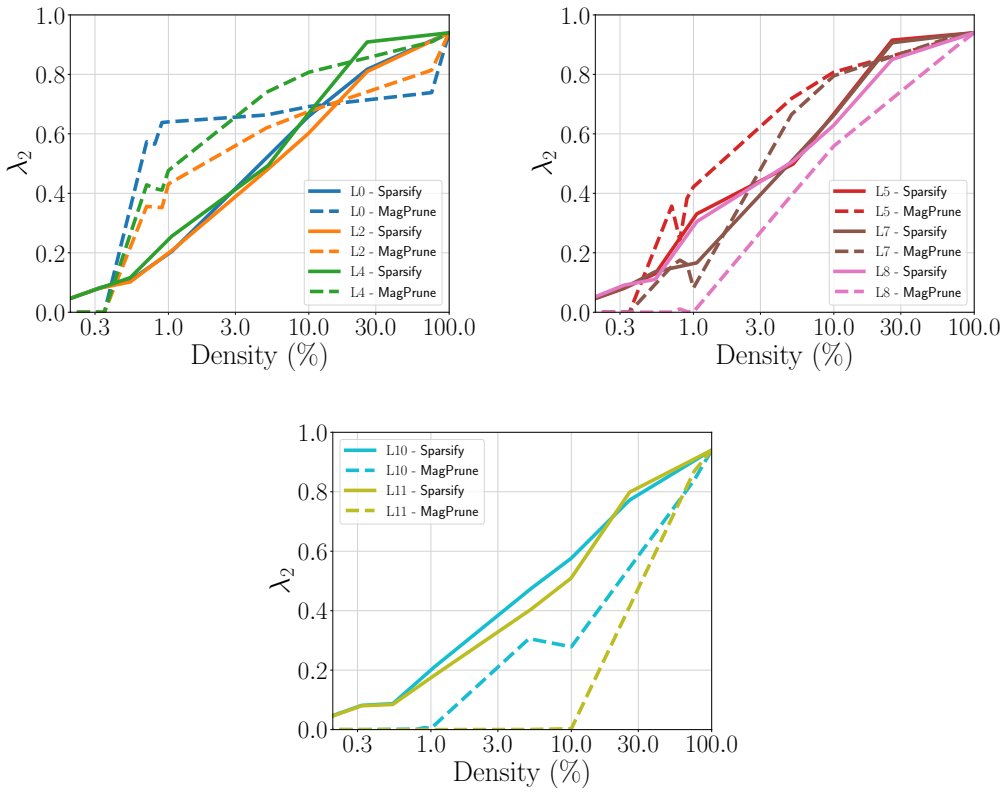

Figure 4. Additional visualisation of second eigenvalue of the Laplacian λ_2 in VGG-11 trained on CIFAR10. See Section 4 for more detail.