Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

Enhancing Accuracy in Deep Learning Using Random Matrix Theory

Leonid Berlyand[1], Etienne Sandier[2], Yitzchak Shmalo[1], and Lei Zhang * [3]

[1]Department of Mathematics, Pennsylvania State University, University Park, PA 16802, USA.
[2]LAMA-CNRS UMR 8050, Université Paris-Est Créteil, Créteil 94010, France.
[3]Institute of Natural Sciences, Shanghai Jiao Tong University, Shanghai 200240, P.R. China.

Abstract. We explore the applications of random matrix theory (RMT) in the training of deep neural networks (DNNs), focusing on layer pruning that reduces the number of DNN parameters (weights). Our numerical results show that this pruning leads to a drastic reduction of parameters while not reducing the accuracy of DNNs and convolutional neural network (CNNs). Moreover, pruning the fully connected DNNs actually increases the accuracy and decreases the variance for random initializations. Our numerics indicate that this enhancement in accuracy is due to the simplification of the loss landscape. We next provide rigorous mathematical underpinning of these numerical results by proving the RMT-based Pruning Theorem. Our results offer valuable insights into the practical application of RMT for the creation of more efficient and accurate deep-learning models.

Keywords:
Deep learning,
Marchenko-Pastur distribution,
Random matrix theory,
Increasing accuracy,
Pruning

Article Info.:
Volume: X
Number: X
Pages: 1 - 66
Date: /2024
doi.org/10.4208/jml.231220

Article History:
Received: 20/12/2023
Accepted: 29/08/2024

Communicated by:
Zhi-Qin John Xu

Contents

*Corresponding author. lzhang2012@sjtu.edu.cn

1 Introduction

Deep neural networks have become a dominant tool for tackling classification tasks, where objects within a set $S \subset \mathbb{R}^n$ are categorized. DNNs are trained on labeled datasets $T \subset \mathbb{R}^n$ by optimizing a loss function such as a cross-entropy loss function in (2.2) to maximize classification accuracy. Through this training process, DNNs have achieved state-of-the-art results on many real-world classification challenges, including handwriting recognition [31], image classification [30], speech recognition [25], and natural language processing [56].

Overfitting is a common challenge for the training of DNNs, which occurs when the model's complexity results in memorization of the training data rather than generalization to new data. Consequently, despite high training set accuracy, the model's performance on the test set deteriorates. To counteract overfitting, different regularization techniques such as dropout [53], early stopping [47], and weight decay regularization [43] have been developed.

Recently, random matrix theory (RMT) has been used in deep learning for addressing overfitting [34, 37]. Similar works have used RMT to obtain RMT-based stopping criteria, see [39], and regularization, see [60]. It has also been shown that RMT can be used to predict DNN performance without access to the test set, see [36, 38], and in general to study the spectrum of weight layers [57] and the input-output Jacobina matrix [45, 46]. RMT-based initializations were also studied in [50]. However, these RMT works in deep learning focused on issues other than utilizing RMT-based pruning in DNNs, which is the focus of our work. Specifically, we study the applications of RMT for pruning DNNs during training. We present numerical simulations on simple DNN models trained on the MNIST, Fashion MNIST, and CIFAR-10 datasets. Generally, these RMT techniques can be extended to other DNN types and any fully connected or convolutional layer of pre-trained DNNs to reduce layer parameters while preserving or enhancing accuracy.

We chose the MNIST, Fashion MNIST, and CIFAR-10 datasets because they balance complexity and efficiency, allowing us to conduct numerous experiments on very large DNNs, which is necessary for assessing the overall behavior of our algorithm within the RMT framework. MNIST is the simplest of these datasets and is easy to train on, while Fashion MNIST and CIFAR-10 are a little more complex. All of these datasets are complex enough to demonstrate the effectiveness of our MP-based pruning method yet simple enough to enable quick and extensive experimentation.

Our RMT pruning approach simplifies DNNs, enabling them to find deeper minima on the loss landscape of the training set. As a result, DNNs can achieve higher accuracy directly on the training set. DNNs, during their training, navigate a complex, multi-dimensional loss landscape in search of the global minimum – the optimal solution. However, the nature of these landscapes can often be rugged, filled with numerous sub-optimal local minima that trap the learning process. By implementing RMT pruning, the landscape becomes smoother, less prone to local minima, and more navigable for the learning algorithm. This makes the optimization process more efficient and enables the DNN to find deeper minima for the loss of the training set.

The works [3,11,61–63] utilized singular value decomposition (SVD) to eliminate small singular values from DNN weight matrices. This pruning of singular values was used to prune the parameters in the weight layer matrices of the DNNs, similar to our work. This pruning was based on techniques such as energy ratio thresholds and monitoring the error of a validation set. However, this energy threshold comes from empirical observations. In contrast, the Marchenko-Pastur threshold used for pruning in our work is justified theoretically by RMT and applied to fully connected networks and simple convolutional neural network (CNNs) to establish an agreement between theory and numerics.

Other pruning methods can be found in [58], in which the authors categorize over 150 studies into three pruning categories: methods that use magnitude-based pruning, methods that utilize clustering to identify redundancy, and methods that use sensitivity analysis to assess the effect of pruning. Our work is mostly related to the first method of pruning; we use MP-based pruning to prune small singular values, together with sparsification to prune all weights of the DNN bigger than some threshold (and set them to 0), see Sections 3.2.1-3.2.3. To the best of our knowledge, we are the first to use the MP distribution as a threshold for pruning. Furthermore, the Pruning Theorem 4.1 provides mathematical justification for the MP-based pruning approach, while Lemma 4.2 provides mathematical justification for the sparsification approach. Other pruning methods, such as pruning at initialization, have also been used, see [48].

In [54], the MP distribution was used to decrease the size of large singular values, which allows for the extraction of the denoised matrix from the original (noisy) one. Then, a validation set was used to determine the SVD pruning threshold for filtering noisy data in DNN classification. However, there are important distinctions with our work. First, we use an MP threshold for directly pruning weights without access to a validation set. Second, the pruning in [54] is done after training, whereas pruning in our work is done during training, corresponding to a different improvement mechanism of accuracy improvement via simplification of loss landscape during training. Third, the authors of [54] focused on how pruning small singular values can improve the accuracy of DNNs trained on noisy data. On the other hand, we study how the pruning of small singular values can guide the pruning of weights of the DNN that are random due to initialization. Moreover, our theoretical approach applies to both sources of randomness: initialization of weights and noise in the data.

In contrast with the above numerical works, our work also provides rigorous theoretical underpinning on the relation between RMT-based pruning of DNN weight matrices and accuracy. To this end, we establish rigorous mathematical results (the Pruning Theorem), which explain the effectiveness of our RMT-based algorithm. The theoretical results will help elucidate the underlying mechanisms of the numerical algorithm, demonstrating why it successfully reduces the number of parameters in a DNN without reducing accuracy. These theoretical results will allow for the development of RMT-based pruning for state-of-the-art DNNs such as ResNets and VITs.

The remainder of this paper is organized as follows. In Section 2, we present an overview of DNN training. In Section 3, we present the numerical results of this paper. In Section 4, we present the Pruning Theorem.

2 Background on deep learning

DNNs have become a widely-used method for addressing classification problems, in which a collection of objects $S \subset \mathbb{R}^n$ is assigned to one of K classes. The objective is to approximate an exact classifier ψ, which maps an element $s \in S \subset \mathbb{R}^n$ to a probability vector $(p_1(s), \ldots, p_K(s))$. In this vector, $p_{i(s)} = 1$ and $p_j = 0$ for $j \neq i(s)$, where $i(s)$ denotes the correct class for s. The exact classifier ψ is known only for a training set T, and DNNs are trained to approximate ψ by constructing a parameterized classifier $\phi(s, \alpha)$ with the aim of extending ψ from T to all of S via $\phi(s, \alpha)$.

This is accomplished by finding parameters α that allow $\phi(s, \alpha)$ to map $s \in T$ to the same class as ψ while maintaining the classifier's ability to generalize to elements $s \in S$. The parameters α are optimized by minimizing a loss function, aiming to enhance the accuracy as the loss declines.

In this study, a DNN is represented as a composition of two functions: the softmax function ρ and an intermediate function $X(\cdot, \alpha)$. The function $X(\cdot, \alpha)$ is defined as a composition of affine transformations and nonlinear activations, as follows:

- $M_l(\cdot, \alpha_l)$ is an affine function that maps $\mathbb{R}^{N_{l-1}}$ to $\mathbb{R}^{N_l}$, and depends on a parameter matrix W_l of size $N_l \times N_{l-1}$ and a bias vector β_l.
- $\lambda : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a nonlinear activation function.
- $X(\cdot, \alpha) = \lambda \circ M_k \cdots \lambda \circ M_1$, where k is the number of layers in the DNN. Note that each λ here might be different from the others, given that the domains of each differ.

Lastly, ρ is the softmax function, which normalizes the output of $X(\cdot, \alpha)$ into probabilities. The components of ρ are calculated as

$$\rho_i(s, \alpha) = \frac{\exp\big(X_i(s, \alpha)\big)}{\sum_{i=1}^{K} \exp\big(X_i(s, \alpha)\big)}. \tag{2.1}$$

The DNN's output, ϕ, is a vector representing the probabilities of an object $s \in S$ belonging to a particular class i. $\phi = \phi(s, \alpha)$, where $\alpha \in \mathbb{R}^\nu$ is the DNN's parameter space and $\nu \gg 1$ is the dimension of the parameter space. The goal is to train the DNN ϕ to approximate the exact classifier by minimizing a loss function, such as the cross-entropy loss function

$$\bar{L}(\alpha) = -\frac{1}{|T|} \sum_{s \in T} \log\big(p_{i(s)}(s, \alpha)\big). \tag{2.2}$$

Training a DNN essentially involves traversing a high-dimensional, non-convex loss landscape to locate the global minimum. But the complexity of these landscapes frequently leads to local minima, saddle points, or flat regions, all of which trap the learning process, impeding it from reaching an optimal solution [16]. These issues amplify as the dimensionality (and thus the complexity) of the DNN increases [13]. In local minima, the gradient of the loss function equals zero, but it is not the global minimum, thus, the algorithm incorrectly assumes it has found the best possible solution. Saddle points, on the

other hand, are points where the gradient is zero, but they are neither a global nor a local minimum. They are particularly problematic in high-dimensional spaces, a common feature in deep learning.

To overcome these obstacles, various sophisticated optimization techniques are employed. For example, optimization algorithms such as Momentum, RMSProp, or Adam are designed to prevent getting stuck by adding additional components to the update rule, which can help in navigating the complex optimization landscape. These methods imbue the optimization process with a form of "memory" of previous gradients, enabling it to continue its search even in flat regions, hence helping to escape local minima and saddle points.

DNNs, with their intricate and numerous parameters, offer formidable modeling capabilities [23]. However, the same attribute that enables their power can also serve as a curse during the training process. Theoretically, DNNs, due to their extensive parameterization, should reach high levels of accuracy on their training sets [64]. But in practice, the accuracy on the training set can often plateau, suggesting the DNN is getting stuck in a local minimum or saddle point of the loss function [20]. This forms a critical impediment, limiting the achievable accuracy on both the training and test sets [27].

Thus, despite the vast number of parameters, DNNs can often find themselves stranded in areas of poor performance. This seemingly paradoxical occurrence is attributable to the interplay between the DNN's architecture, the data it is training on, and the optimization process being employed [23]. Factors like poor initialization, inappropriate learning rates, or the vanishing/exploding gradients problem can cause the DNN to settle in sub-optimal regions of the loss landscape [21].

Techniques like gradient clipping [44] can aid in overcoming these issues. Regularization techniques, which either penalize complex models or enforce sparsity in the weight matrix, can also assist in avoiding local minima [23]. Nonetheless, these methods do not alter the fundamental structure of the loss landscape, indicating that the problem of local minima remains [20].

The potential of the RMT approach stands out in this context. We show that utilizing RMT in pruning the DNN's weight layers simplifies the loss landscape. This simplification reduces the incidence of local minima and saddle points, aiding the optimization process in its quest for a global minimum. In doing so, the DNN might attain higher levels of accuracy on the training set directly without reaching a plateau. This results in an overall enhancement in model performance, as higher training set accuracy generally translates to improved performance on the test set, assuming overfitting does not occur.

3 Numerical algorithm and experiments

In this section, we focus on the training of two DNNs: The normal DNN, which keeps all of its singular values, and a pruned DNN based on Algorithm 1, see Section 3.1.2. Each DNN is trained for a predetermined number of epochs, with the number of epochs varying per example. The DNNs are also trained for multiple seeds to ensure the reproducibility of the simulations.

The performance of the DNNs is evaluated by plotting the average accuracy and variance of accuracy for the different seeds. This allows us to visually compare the performance of both the normal and pruned DNNs. For more numerical results using a slightly different RMT training approach, see [52].

3.1 Numerical algorithm

3.1.1 An overview of the Marchenko-Pastur (MP) distribution and its applications in machine learning

We start with the MP distribution from RMT. This distribution is of fundamental importance in RMT and has numerous applications, such as signal processing, wireless communications, and machine learning, as described in [14, 19, 51, 59]. The MP distribution characterizes the limiting spectral density of large random matrices and conveys information about the asymptotic distribution of eigenvalues in a random matrix, predicting the behavior of random matrices under various conditions. Additionally, the MP distribution is utilized in principal component analysis (PCA) and other dimension reduction techniques, see [1, 10, 49].

To begin, we introduce the empirical spectral distribution (ESD) of an $N \times M$ matrix G as follows.

Definition 3.1. *The ESD of an $N \times M$ matrix G is given by*

$$\mu_{G_M} = \frac{1}{M} \sum_{i=1}^{M} \delta_{\sigma_i}, \tag{3.1}$$

where σ_i denotes the i-th non-zero singular values of G, and δ represents the Dirac measure.

Theorem 3.1 ([35])**.** *Let W be an $N \times M$ random matrix with $M \leq N$. The entries $W_{i,j}$ are independent and identically distributed random variables with mean 0 and variance $\sigma^2 < \infty$. Define $X = W^\top W/N$. Assuming that $N \to \infty$ and $M/N \to c \in (0, +\infty)$, the ESD of X, denoted by μ_{X_M}, converges weakly in distribution to the Marchenko-Pastur probability distribution*

$$\frac{1}{2\pi\sigma^2} \frac{\sqrt{(\lambda_+ - x)(x - \lambda_-)}}{cx} \mathbf{1}_{[\lambda_-, \lambda_+]} dx \tag{3.2}$$

with

$$\lambda_\pm = \sigma^2 (1 \pm \sqrt{c})^2. \tag{3.3}$$

This theorem asserts that the eigenvalue distribution of a random matrix converges to the Marchenko-Pastur distribution as its dimensions increase. The MP distribution is a deterministic distribution, dependent on two parameters: the variance of the random variables in the initial matrix σ^2, and the ratio of the number of columns to the number of rows c.

3.1.2 Using MP for pruning DNN weights

As stated in Section 2, a DNN is a composition of affine functions M_l and non-linear activation functions. The affine functions M_l can be thought of as a $N \times M$ matrix W_l of parameters and a bias vector β_l. In this work, we only focus on the matrix W_l of parameters. It has been shown that W_l can be studied using the spiked model approach in random matrices, with the ESD of $X_l = W_l^\top W_l/N$ having some eigenvalues which are bigger than λ_+ and some eigenvalues which are smaller than λ_+, see [37,54]. In this paper, we focus on weight layer matrices W_l which were initialized in such a way that $\sqrt{N}W_l(0)$ satisfy the assumptions of W given in Theorem 3.1. Then, we look at the ESD of $B_l = W_l^\top W_l$, without normalizing by $1/N$. This setting is more applicable for the situation in which the components of R are i.i.d. taken from $N(0, 1/N)$.

Thus, we take $B_l(t) = W_l(t)^\top W_l(t)$ with $W_l(t)$ a $N \times M$ weight of the l-th layer matrix at time t of DNN training. We use RMT to study the deformed matrix $W_l(t) = R_l(t) + S_l(t)$ with $R_l(t)$ random and $S_l(t)$ a deterministic matrix. One can assume that during training we go from $W_l(0) = R_l$ (i.e. W_l is random) to $W_l(t_{\text{final}}) = R_l(t_{\text{final}}) + S_l(t_{\text{final}})$ with $\|S_l(t_{\text{final}})\| \neq 0$ and t_{final} the final training time. Meaning that as $t \to t_{\text{final}}$, $\|S_l(t)\|$ grows and so $W_l(t)$ becomes less random.

An important question is: Why does training reduce randomness in weight matrices?

- Suppose a DNN has only one weight layer matrix W. Before training starts, the matrix $W(0)$ is initialized with DNN weights. Entries of W arranged into a vector $\alpha(0)$ are chosen randomly, meaning $W(0)$ is fully random. A gradient descent step can be written as
$$\alpha(n+1) = \alpha(n) - \tau \nabla L\big(\alpha(n)\big). \tag{3.4}$$
The loss gradient $\nabla L(\alpha(n))$ is determined by the training data T, which is mostly deterministic. That is, when we take a step from $n = 0$ to $n = 1, \alpha(0)$, the random DNN parameters are gradually replaced with deterministic parameters, and this process continues throughout training.

- However, T is only mostly deterministic. Each object $s \in T$ is sampled from a probability distribution and is a random variable, so T contains some randomness.

- In practice, the randomness of $\alpha(n)$ decreases as $n \to \infty$ ($\|S(n)\|$ increases), but some randomness due to the data remains.

We make the following observations based on the singular values of the weight layer matrix W_l:

- Observation 1: Singular values σ_i of W_l that are smaller than a threshold $\sqrt{\lambda_+}$ are likely to be singular values of R_l, where λ_+ is the upper bound of the MP distribution of $R^\top R$. For more on this observation, see [54,57].

- Observation 2: R_l does not enhance the accuracy of a DNN. In other words, the random components of the weight layers do not contain any valuable information and, therefore do not improve accuracy, see Lemma 4.2.

Based on these observations, the main idea is to remove some randomness from the DNN by eliminating some singular values of W_l smaller than the threshold $\sqrt{\lambda_+}$. Algorithm 1 describes this procedure.

Algorithm 1 Optimized DNN Training and Pruning for Parameter Efficiency.

Require: ℓ, a predetermined number of epochs; τ, a threshold for the MP fit criteria in Section D.4; $f(\text{epoch})$, a monotonically decreasing function from 1 to 0 (i.e. (3.5)) and for each $1 \leq l \leq L$ and weight layer matrix W_l **state** $split_l = false$.

1: Initialize: Train the DNN for ℓ epochs. Take epoch := ℓ.
2: **while** a predefined training condition is met (i.e. epoch ≤ 100) **do**
3: **for** each l, if $split_l = false$ then for weight matrix W_l in the DNN ϕ **do**
4: Perform SVD on W_l to obtain $W_l = U_l \Sigma_l V_l^\top$.
5: Calculate eigenvalues of $W_l^\top W_l$.
6: Apply BEMA algorithm (see Section D.1) to find the best fit MP distribution for ESD of $X = W_l^\top W_l$ and corresponding λ_+.
7: Check if ESD of X fits the MP distribution using MP fit criteria from Section D.4 and threshold τ.
8: **if** ESD fits the MP distribution **then**
9: Eliminate the portion $(1 - f(\text{epoch}))$ of singular values smaller than $\sqrt{\lambda_+}$ to obtain Σ' and form $W_l' = U_l \Sigma_l' V_l^\top$.
10: Use Σ' to create $W_{1,l}' = U_l \sqrt{\Sigma_l'}$ and $W_{2,l}' = \sqrt{\Sigma_l'} V_l^\top$.
11: **if** $W_{1,l}'$ and $W_{2,l}'$ together have fewer parameters than W_l' **then**
12: Replace W_l in the DNN ϕ with $W_{1,l}' W_{2,l}'$, change $split_l = true$.
13: **else**
14: Replace W_l in the DNN ϕ with W_l'.
15: **end if**
16: **else**
17: Do not replace W_l.
18: **end if**
19: **end for**
20: Train the DNN for ℓ epochs. Take epoch := epoch + ℓ.
21: **for** each l, if $split_l = true$ **do**
22: **if** for $W_l := W_{1,l}' W_{2,l}'$ the ESD of X_l fits the MP distribution with thresholds τ and λ_+ **and** if, we (hypothetically) applied steps 4-12 to W_l, the number of parameters in the DNN ϕ would decrease **then**
23: replace $W_{1,l}' W_{2,l}'$ with W_l and $split_l = false$.
24: **else**
25: Do not change anything.
26: **end if**
27: **end for**
28: **end while**

3.1.3 MP and Tracy-Widom distribution for DNN training

We use the bulk eigenvalue matching analysis (BEMA) algorithm (see Section D.1) to find the MP distribution that best fits the ESD of $X_l = W_l^\top W_l$, with W_l a weight layer matrix. We then use the Tracy Widom distribution (see [28]) to find a confidence interval for the λ_+ of the ESD of X_l and then prune the small singular values of W_l based on the MP-based threshold $\sqrt{\lambda_+}$, see Section D.1 for more details on the Tracy-Widom distribution and the BEMA algorithm. The steps of this procedure are shown in Algorithm 1.

In step 9 of Algorithm 1, we ensure not to eliminate all of the small singular values (i.e. singular values whose corresponding eigenvalues fall within the MP distribution). Striking a balance between removing the smaller singular values and retaining some is found to be essential. Removing all of the smaller singular values might result in the underfitting of the DNN, thereby inhibiting its learning capability. Conversely, retaining some of the smaller singular values adds a degree of randomness in the weight layer matrix W_l, which is found to impact the DNN's performance positively.

Remark 3.1. Note that it is possible to continue splitting the matrices W_ℓ, W'_ℓ, and so on. This also improves accuracy for fully connected layers. However, we found that recombining and splitting the original matrix works better.

3.2 Numerical experiments

Numerical simulations presented in this paper show that MP-based pruning enhances the accuracy of DNNs while reducing the number of DNN weights (parameters)[1]. The first set of numerical simulations employs fully connected DNNs trained on the MNIST and Fashion MNIST datasets, revealing that MP-based pruning during training improves accuracy by 20-30% while reducing the parameter count by 30-50%. These findings are consistent across various architectures and weight initializations, underscoring the consistency of the MP-based pruning approach. Further, the combination of this approach and sparsification (eliminating parameters below a certain threshold, see [58]) leads to even more significant reductions in parameters (up to 99.8%) while increasing accuracy (by 20-30%). This reduction in parameters is greater than what is achievable through sparsification alone (99.5%), see Section 3.2.1.

Unless stated otherwise, in all numerical simulations, the parameter matrices were initialized from $N(0, 1/N)$, with N the number of input features, while the bias vectors were initialized to 0. This initialization is closest to the theoretical work in this paper, which is why we use it. The ReLU activation function was applied after every layer, including the final layer. While it might be easier to train DNNs with other initializations and architectures, we found that we obtained the highest accuracies when training with the affirmation structure while using MP-based pruning. For example, using a fully connected DNN, we obtained a 91.27% accuracy on the Fashion MNIST test set, which is the highest accuracy we observed on the data set using a fully connected DNN (see Section 3.2.1). MP-based pruning also increases the accuracy of fully connected DNNs that do not have an activation function on the final layer, for example, see Section E.3.

[1] Code can be found at https://github.com/yspennstate/RMT_pruning_2/blob/main/rmt_pruning_2.ipynb

For simplicity of presentation, we choose to demonstrate the MP-based pruning for fully connected DNNs. In short, the idea is as follows. First, we observe that weight layer matrices W_l have singular values of two types: those that contain information and the ones that do not and, therefore, can be removed (pruned). This separation is done via MP threshold $\sqrt{\lambda_+}$. Furthermore, we demonstrate that pruning based on this MP threshold preserves DNN accuracy. These numerical findings are supported by rigorous mathematical results (Theorem 4.2). In fact, for the case of fully connected layers we show numerically that MP-based pruning simplifies the loss landscape, leading to a significant increase in DNN accuracy (by 20-30%). Finally, we show that a combination of MP-based pruning with sparsification preserves or even increases accuracy while reducing parameters by 99.8% vs. MP-based pruning alone, with a reduction of 30-50%, or sparsification alone, with a reduction of 99.5%. Our theoretical results also explain why sparsification does not reduce accuracy; see Lemma 4.2 and Remark 4.3.

Our numerics explores the application of MP-based pruning on DNNs that already achieve relatively high accuracy on MNIST (Section 3.2.1), Fashion MNIST (Section 3.2.1), and CIFAR10 datasets (Section 3.2.3), including those using convolutional neural networks and sparsification techniques (Sections 3.2.1-3.2.3). Our results show a substantial reduction in parameters (over 95%) while preserving accuracy through a combination of MP-based pruning during training and post-training sparsification, surpassing the efficiency of using sparsification alone (80-90% reduction in parameters). These extensive simulations for various architectures and initializations demonstrate the consistency and wide applicability of MP-based pruning in optimizing DNN performance.

To see how MP-based pruning performed on a simple regression problem, see Section E.3.

Training and testing procedure

The training and testing procedure for each network consists of the following steps:

1. We follow the standard partition for MNIST and Fashion MNIST, with 60, 000 images for training and 10, 000 images for testing.
2. We train the network for a certain number of epochs.
3. We test the network after each epoch and store the accuracy for later comparison.
4. For the pruned DNN, we apply Algorithm 1 after a set number of epochs (defined by the split frequency).

3.2.1 Training of fully connected DNNs on MNIST: Simplifying the loss landscape

The results of the simulations are presented in the examples below. These examples show figures and tables which compare the average accuracy and variance of accuracy for both the normally trained and pruned DNNs, as well as the average loss and number of parameters in both DNNs. Detailed discussion and analysis of these results are presented in the examples.

Training hyperparameters:

- Split frequency (ℓ) (every how many epochs we split the pruned DNN and remove small singular values): 7.
- Goodness of fit (GoF or τ) = 0.7.

See Section E.5 for the other hyperparameters in these simulations.

Remark 3.2. The algorithm for finding the GoF parameter is given in Section D.4. It is used to determine if the assumption given in Theorem 4.1, that $W_l = R_l + S_l$, is reasonable and that the weight layers can reasonably be modeled as a spiked model (that is W_l is a deformed matrix).

Example 3.1. We conducted several numerical simulations to compare the performance of the normal DNNs, trained using conventional methods, and pruned DNNs, trained using our RMT approach.

In all simulations, the networks start with different initial topologies and are trained over a course of 40 epochs. The portion of singular values smaller than $\sqrt{\lambda_+}$ that we retain (see step 3 in Algorithm 1) is given by the linear function

$$f(\text{epoch}) = \max\left(0, -\frac{1}{30} \cdot \text{epoch} + 1\right). \tag{3.5}$$

The topologies and the results of the simulations are summarized in Table 3.1 and Fig. 3.1. The results indicate a consistent trend across different topologies: The pruned DNNs outperform the normal DNNs in terms of accuracy on the test set while also displaying smaller variance across multiple runs. Furthermore, the pruned DNNs consistently achieve a significant reduction in parameters by the end of the training, see Remark 3.4.

Remark 3.3. In these examples, the goodness of fit parameter can be very large (even 1) and does not change the accuracy of the DNN. This is not always the case, especially for state-of-the-art pre-trained DNNs, as we will show in future works. See also Section 3.2.2 for an example of when GoF must be smaller.

Remark 3.4. In Table 3.2, we observe the effect of our RMT training approach on the number of parameters in our DNN with different topologies. Each topology started with

Table 3.1: Performance of normal and pruned DNNs for different initial topologies.

Initial topology	Unpruned DNN accuracy	Pruned DNN accuracy
[784, 3000, 3000, 2000, 500, 10]	∼85%	∼98.5%
[784, 1000, 1000, 1000, 500, 10]	∼70%	∼98.5%
[784, 2000, 2000, 1000, 500, 10]	∼70%	∼98.5%
[784, 1500, 3000, 1500, 500, 10]	∼70%	∼98.5%
[784, 1000, 1000, 1000, 500, 10] (GoF 1)	∼70%	>98.5%

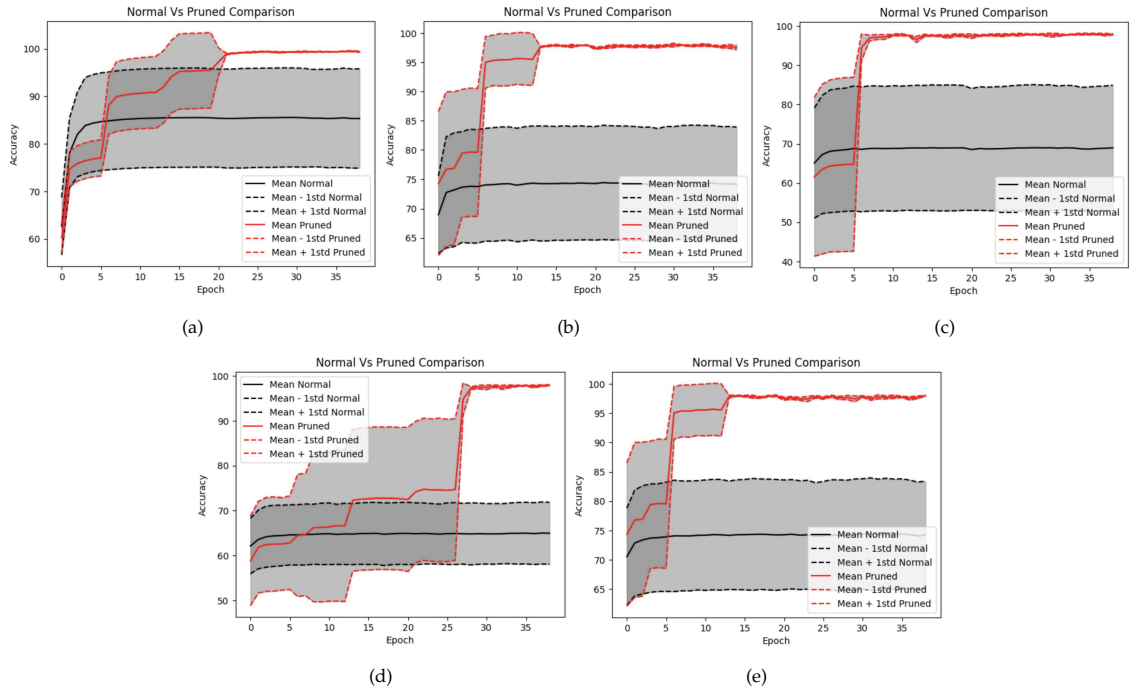


Figure 3.1: Comparison of Normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the test set. The sub-figures correspond to the different initial topologies: (a) $[784, 3000, 3000, 2000, 500, 10]$, (b) $[784, 1000, 1000, 1000, 500, 10]$, (c) $[784, 2000, 2000, 1000, 500, 10]$, (d) $[784, 1500, 3000, 1500, 500, 10]$, and (e) $[784, 1000, 1000, 1000, 500, 10]$ with a larger goodness of fit parameter of 1.

Table 3.2: DNN topology, initial and final parameters for the pruned DNNs, with percentage reductions.

Topology	Initial parameters	Final parameters	Percentage reduction
$[784, 3000, 3000, 2000, 500, 10]$	18,365,510	10,471,510	42.98%
$[784, 1000, 1000, 1000, 500, 10]$	3,292,510	1,678,234	49.03%
$[784, 2000, 2000, 1000, 500, 10]$	8,078,510	4,619,376	42.82%
$[784, 1500, 3000, 1500, 500, 10]$	10,937,510	5,554,966	49.21%

a fixed number of parameters, and by the end of training, we see a significant reduction in the number of parameters across all topologies for the pruned DNN. For each topology and across all seeds, the reduction in the number of parameters was consistent, indicating the robustness of our training process in pruning the network while maintaining performance.

Simplification of loss landscape for more efficient training. As mentioned, a common challenge with DNNs is the complex and high-dimensional loss landscape due to the large number of parameters. This complexity often leads to local minima or saddle points that hinder optimal training. However, by using this clever RMT pruning approach, we effectively eliminate redundant parameters, thereby simplifying the loss landscape. This

simplification allows us to avoid suboptimal local minima and converge more readily to a global minimum.

This improved optimization efficiency is evident when comparing the loss and accuracy of the original and pruned DNNs on both training and test sets; see Table 3.3 and Fig. 3.2. The pruned DNNs achieve lower loss and higher accuracy on the training set directly, indicating that they are finding deeper minima in the loss landscape and avoid suboptimal local minima.

Thus, the RMT pruning approach not only significantly reduces the complexity of DNNs but also enhances their performance by improving their optimization efficiency. Despite the reduction in parameters, the pruned DNNs still exhibit excellent performance on both training and test sets (even higher accuracy than the normally trained DNNs), demonstrating the effectiveness of this approach.

Simplifying the loss landscape for fully connected DNNs on Fashion MNIST. In this section, we trained the normal and pruned DNNs on the data set Fashion MNIST and

Table 3.3: Comparison of training and test losses between normal and pruned DNNs for different topologies.

Topology	Type	Training loss	Test loss
$[784, 3000, 3000, 2000, 500, 10]$	Normal	1.324090	150.541977
	Pruned	0.015084	9.211179
$[784, 1000, 1000, 1000, 500, 10]$	Normal	0.938041	127.608980
	Pruned	0.001350	14.543617
$[784, 2000, 2000, 1000, 500, 10]$	Normal	0.567025	70.037265
	Pruned	0.019599	13.676276
$[784, 1500, 3000, 1500, 500, 10]$	Normal	1.037013	146.353504
	Pruned	0.009206	9.519642

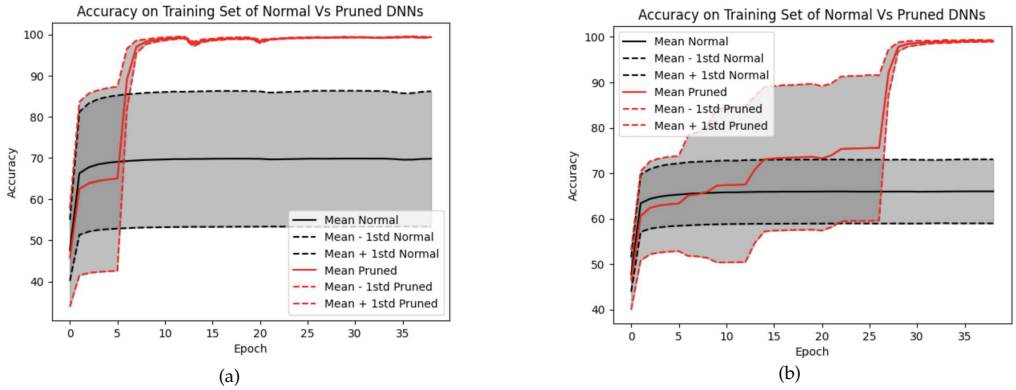


Figure 3.2: Comparison of normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the training set. The sub-figures correspond to the different initial topologies: (a) $[784, 2000, 2000, 1000, 500, 10]$, (b) $[784, 1500, 3000, 1500, 500, 10]$. The other examples in Fig. 3.1 have similar-looking accuracies on their training set.

we look at the performance of both DNNs on the training and test set. Again the pruned DNN obtains higher accuracy and lower loss on both the training and test sets, evidence that pruning the DNN using RMT simplifies the loss landscape and allows the DNN to find a deeper global minimum.

Training hyperparameters:

- Split frequency (every how many epochs we split the modified DNN and remove small singular values): 7.
- Goodness of fit = 0.7.

The other hyperparameters for the simulations in this subsection can be found in Section E.6.

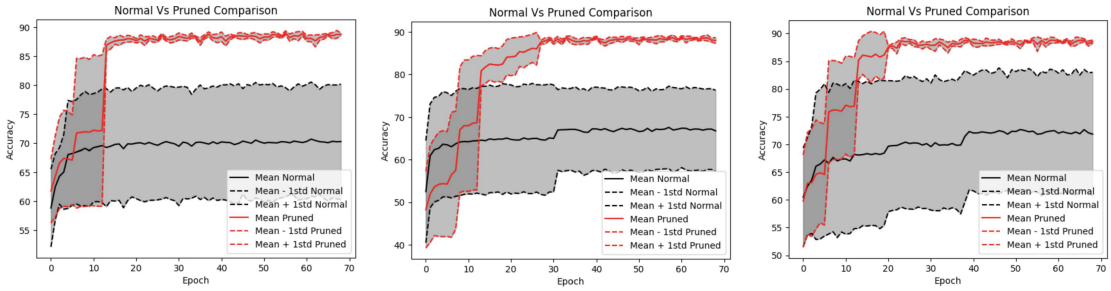
In all simulations, the networks start with different initial topologies, are trained over a course of 70 epochs, and the portion of singular values smaller than $\sqrt{\lambda_+}$ that we retain is given by the linear function

$$f(\text{epoch}) = \max\left(0, -\frac{1}{60} \cdot \text{epoch} + 1\right). \tag{3.6}$$

Example 3.2. The topologies and the results of the simulations are summarized in Table 3.4 and Fig. 3.3.

As with MNIST, in the case of training on Fashion MNIST the results indicate a consistent trend across different topologies: the pruned DNNs outperform the normal DNNs in terms of accuracy on the test set while also displaying smaller variance across multiple runs. Furthermore, the pruned DNNs consistently achieve a significant reduction in parameters by the end of the training, see Table 3.5.

In Table 3.5, we observe the effect of our RMT training approach on the number of parameters in our DNN with different topologies. Each topology started with a fixed number of parameters, and by the end of training, we see a significant reduction in the number of parameters across all topologies for the pruned DNN. For each topology and across all seeds, the reduction in the number of parameters was consistent, indicating the robustness of our training process in pruning the network while maintaining performance.



(a) (b) (c)

Figure 3.3: Comparison of normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the test set. The sub-figures correspond to the different initial topologies: (a) $[784, 2000, 4000, 2000, 500, 10]$, (b) $[784, 2000, 2000, 2000, 2000, 1000, 500, 10]$, (c) $[784, 3000, 4000, 3000, 500, 10]$.

Table 3.4: Performance of normal and pruned DNNs for different initial topologies.

Initial topology	Unpruned DNN accuracy	Pruned DNN accuracy
$[784, 2000, 4000, 2000, 500, 10]$	∼70%	∼89%
$[784, 2000, 2000, 2000, 2000, 1000, 500, 10]$	∼65%	∼89%
$[784, 3000, 4000, 3000, 500, 10]$	∼70%	∼89%

Table 3.5: DNN topology, initial and final parameters for the pruned DNNs, with percentage reductions.

Topology	Initial parameters	Final parameters	% Reduction
$[784, 2000, 4000, 2000, 500, 10]$	18,581,510	10,471,510	43.65%
$[784, 2000, 2000, 2000, 2000, 1000, 500, 10]$	16,082,510	8,950,860	44.34%
$[784, 3000, 4000, 3000, 500, 10]$	27,867,510	15,599,740	44.02%

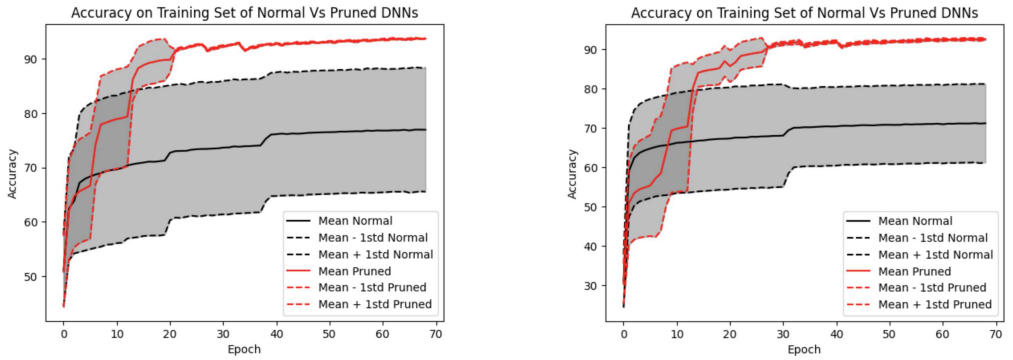
Remark 3.5. Again, we see that the RMT approach helps simplify the loss landscape so that during gradient descent the pruned DNN finds a deeper global minimum than the normal DNN. We can see this by looking at the accuracy of the DNNs on the training set, see Fig. 3.4.

One can also see that the pruned DNN is obtaining a deeper global minimum by looking at Table 3.6.

We present two graphs to analyze the impact of pruning on the performance of the DNN with architecture $[784, 2000, 4000, 2000, 500, 10]$, see Fig. 3.5. Fig. 3.5(a) shows the training and testing accuracy of the DNN over epochs. The blue dashed lines indicate the points at which pruning was applied. It is observed that the accuracy does not change significantly after pruning, especially during the initial epochs when the DNN parameters are still random. Fig. 3.5(b) specifically examines the impact of pruning on training accuracy. Red dots represent the accuracy before pruning, and purple dots represent the accuracy after pruning. The graph demonstrates that the training accuracy remains relatively stable before and after pruning, reinforcing the observation that MP-based pruning does not drastically affect performance, particularly in the early stages of training. After the pruning, the training seems to be easier, and the DNN accuracy improves as the training continues.

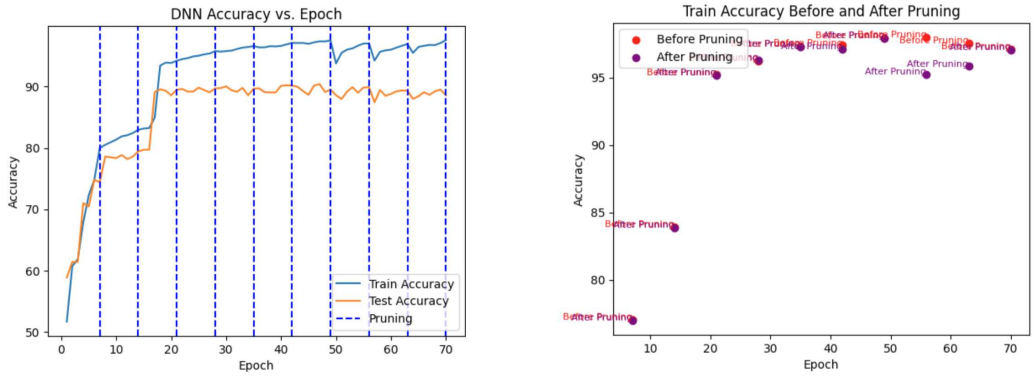
Table 3.6: Comparison of training and test losses between normal and modified DNNs for different topologies.

Topology	Type	Training loss	Test loss
$[784, 2000, 4000, 2000, 500, 10]$	Normal	0.460549	95.516107
	Pruned	0.144334	47.063285
$[784, 3000, 4000, 3000, 500, 10]$	Normal	0.880157	130.785274
	Pruned	0.191681	49.781472
$[784, 2000, 2000, 2000, 2000, 1000, 500, 10]$	Normal	0.790270	127.023410
	Pruned	0.270197	42.925396



(a) (b)

Figure 3.4: Comparison of normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the training set. The sub-figures correspond to the different initial topologies: (a) $[784, 3000, 4000, 3000, 500, 10]$, (b) $[784, 2000, 2000, 2000, 2000, 1000, 500, 10]$.



(a) Accuracy vs. Epoch (b) Train Accuracy Before and After Pruning

Figure 3.5: Analysis of DNN training and pruning.

We applied the MP-based pruning approach for DNNs with other initializations, such as the He and Xavier initializations, and obtained similar results in improvements of accuracy, see Section E.2.

Also, for the DNN with architecture $[784, 2000, 4000, 2000, 500, 10]$, we computed the cost of training the DNN for 70 epochs on the Intel Xeon CPU with 2 vCPUs (virtual CPUs) and 13GB of RAM chip, with MP-based pruning and a split frequency of 7. The total cost was $12,215.13$ seconds, and we achieved an accuracy of $\sim 89.7\%$. Finally, we trained the same DNN for the same amount of CPU time but without MP-based pruning, and the DNN accuracy plateaued at $\sim 84\%$ accuracy. This illustrates that the increase in accuracy provided by MP-based pruning is not obtained because of increases in computational costs alone.

We applied the MP-based pruning approach on a DNN without adding any regularization while training with GD alone and with a fixed $lr = 0.01$. The DNN architecture was $[784, 3000, 3000, 3000, 3000, 500, 10]$ and we trained for 70 epochs with a split frequency

of 7. Without pruning, the accuracy plateaued at 44%, while with pruning, it reached $\sim$ 80%. This indicates that other strategies, such as regularization or rate decay, are useful to take advantage of MP-based pruning. However, MP-based pruning improves the accuracy of GD alone.

We trained a DNN with architecture $[784, 3000, 3000, 3000, 500, 10]$ on Fashion MNIST for 300 epoch, while adding both $L1$ and $L2$ regularization to the loss, see (E.7). The hyperparameters for the regularization were 0.0000005 and 0.0000001, respectively; the split frequency was 13 and the number of singular values kept was given by

$$f(\text{epoch}) = \max\left(0, -\frac{1}{1000} \cdot \text{epoch} + 1\right).$$

All other hyperparameters were kept the same as in Example 3.3. The DNN achieved a 100% accuracy on the training set and a 91.27% accuracy on the test set, showing that the MP-based pruning algorithm attains higher accuracy when we combine it with regularization. More on the relationship between regularization and MP-based pruning will be discussed in another paper.

MP-based pruning with sparsification for fully connected DNNs on Fashion MNIST. We train a fully connected DNN on Fashion MNIST to achieve $\sim$ 89% accuracy (on the test set) with the same MP-based pruning approach as in Section 3.2.1 for a DNN with the topology [784, 3000, 3000, 3000, 3000, 500, 10]. At the end of the training, we employ the sparsification method by setting to zero weights in the DNN smaller than the sparsification threshold ξ. Fig. 3.6 shows the accuracy of the DNN vs. the number of parameters kept (determined by varying ξ). This additional sparsification leads to a large reduction in parameters, by over 99.5%, without a significant drop in accuracy ($\sim$ 0.5% drop). Assuming that the weights of the DNN which are smaller than the threshold ξ are i.i.d. from a distribution with zero mean and bounded variance, Lemma 4.2 provides an explanation

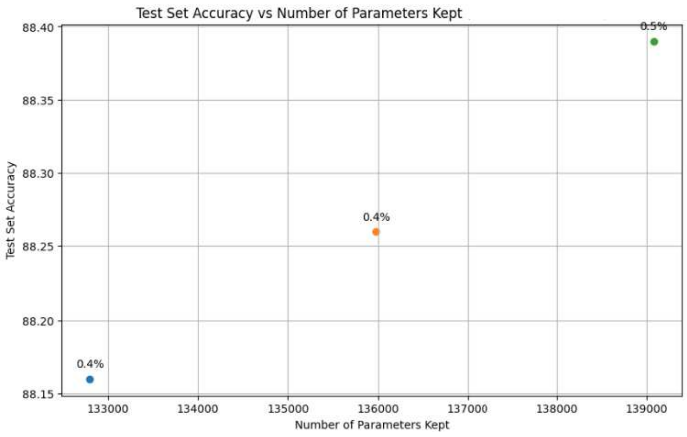


Figure 3.6: Accuracy vs. number of parameters kept. The percentage of parameters kept is also shown (above each point on the graph).

for why removing the small weights (sparsification) does not affect accuracy.

Alternatively, one can prune the weight layers during training by combining the MP-based pruning approach together with sparsification (i.e. removing weights smaller than the threshold ξ every couple of epochs). We performed this training on the above DNN, with ξ depending on the epoch. In our case, we initially took $\xi = 0.001$ and set it to grow linearly so by the end of training $\xi = 0.02$. We achieved a 88% accuracy, while the final DNN had $71,331$ parameters (keeping $\sim 0.2\%$ parameters).

Finally, we tried the sparsification pruning method during training without MP-based pruning. Again, we initially took $\xi = 0.001$ and set it to grow linearly so that by the end of training $\xi = 0.02$. Similar to our observations from Section 3.2.1, the DNN plateaued at $\sim 70\%$ accuracy while having $128,533$ parameters at the end of training (keeping $\sim 0.4\%$ parameters). Thus, we see that a combination of MP-based pruning with sparsification is useful for pruning while also increasing DNN accuracy for fully connected DNNs trained on Fashion MNIST.

3.2.2 MP-based pruning of CNNs on MNIST and Fashion MNIST

In our further exploration, we perform numerical simulations on convolutional neural networks using MNIST and Fashion MNIST. In this simulation, our primary objective is to investigate the effect of pruning the small singular values of the convolutional layers. The overall goal in this example is to reduce the number of parameters in the CNN while at the same time preserving its accuracy.

Given the multidimensional nature of convolutional layers, the direct application of singular value decomposition is not straightforward. To overcome this challenge, we first transform each convolutional layer into a 2-dimensional matrix. Specifically, for a convolutional layer with dimensions $m \times n \times p \times q$ (where m is the number of output channels, n is the number of input channels, and $p \times q$ is the kernel size), we reshape it into a matrix of size $m \times npq$.

After this flattening process, we proceed with the pruning as before, employing SVD to remove the smaller singular values. This step essentially compresses the convolutional layer, reducing its complexity while hopefully maintaining its representational capability.

The hyperparameters for the simulations in the next example can be found in Section E.7. The other parts of the CNN architecture (which is the same for all of the CNNs in this paper) can be found in Section E.8.

The learning rate (lr) is also modified every epoch to be

$$lr_n = lr_{n-1} * 0.96, \tag{3.7}$$

where lr_k is the learning rate at epoch k. Thus it decays over the learning time, see [23] for more information.

Example 3.3. In this first example, we trained a CNN on MNIST for 30 epochs with a split frequency of 13. The convolutional layers are given by $[1, 64, 128, 256, 512]$, and the model starts with one input channel, and then each subsequent number represents the number of filters in each subsequent convolutional layer. Therefore, the model has 4 convolutional

layers with filter sizes of $64, 128, 256$, and 512, respectively. We apply a kernel for each layer of size 3×3.

The fully connected layers are given by $[41472, 20000, 10000, 5000, 3000, 1400, 10]$, we see that the model has 6 fully connected layers. The GoF parameter for the fully connected layers is 0.6 while the GoF parameter for the convolutional layers is 0.05.

In these numerical simulations, the portion of singular values smaller than $\sqrt{\lambda_+}$ that we retain is given by the linear function

$$f(\text{epoch}) = \max\left(0, -\frac{1}{20} \cdot \text{epoch} + 1\right). \tag{3.8}$$

The accuracy of this DNN on the training and test set is given in Fig. 3.7.

In Example 3.3, it was observed that the pruned CNN exhibited a slightly lower accuracy in comparison to the normally trained CNN. Remarkably, despite this marginal drop in performance, the pruned CNN managed to maintain this level of accuracy with approximately half of the parameters used by the normally trained CNN. While the normally trained CNN has $1,100,323,974$ parameters (on account of how large the fully connected layers are), the pruned CNN has $583,670,449$ parameters.

In terms of performance on the training set, the normally trained CNN demonstrated an accuracy of 100%, an indicator of its potential overfitting to the training data. This is in contrast with the pruned CNN, which displayed a lower accuracy on the training set. The narrower gap between the training set and test set accuracies for the pruned CNN could be interpreted as a sign of reduced variance between the training and test set, suggesting less overfitting in the pruned model. At the same time, the fact that the normal CNN archives have high accuracy on the training set suggests that the loss function in this example is simple- i.e. finding the global max of the loss function is simple. This might be why the pruned CNN does not outperform the unpruned CNN.

In this example, the drop in test set accuracy for the pruned CNN is dependent on the GoF parameter. As the GoF parameter becomes more restrictive, the drop in accuracy

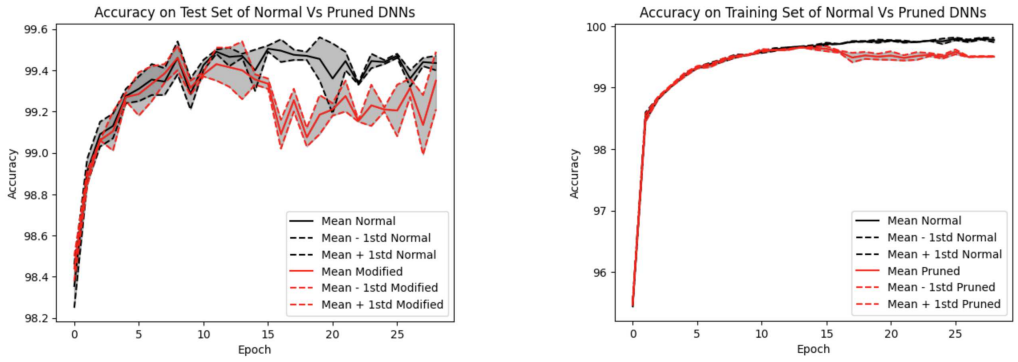


Figure 3.7: Comparison of normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the test and training sets.

becomes less pronounced. However, it is important to note that a more restrictive GoF parameter also leads to a smaller reduction in parameters. These observations suggest a delicate balance between the GoF parameter, model complexity (as indicated by the number of parameters), and model performance.

Example 3.4. In this example, we trained a CNN on the fashion MNIST dataset for 70 epochs with a split frequency of 17. The convolutional layers are given by $[1, 64, 128, 256, 512]$; we again apply a kernel for each layer of size 3×3. The fully connected layers are given by $[41472, 10000, 5000, 5000, 10]$. The GoF parameter for the fully connected layers is 0.7, while the GoF parameter for the convolutional layers is 0.15.

In this numerical simulation, the portion of singular values smaller than $\sqrt{\lambda_+}$ that we retain is given by the linear function

$$f(\text{epoch}) = \max\left(0, -\frac{1}{60} \cdot \text{epoch} + 1\right). \tag{3.9}$$

The accuracy of this DNN on the training and test set is given in Fig. 3.8.

In Example 3.4, the pruned CNN exhibited a slightly lower accuracy in comparison to the conventionally trained CNN. Despite this marginal drop in performance, the pruned CNN maintained this level of accuracy with approximately half of the parameters used by the conventionally trained CNN. While the normally trained CNN had $491,381,774$ parameters, the pruned CNN utilized only $261,891,332$ parameters.

In terms of performance on the training set, the conventionally trained DNN demonstrated an accuracy of approximately 99%, suggesting potential overfitting to the training data. On the other hand, the pruned DNN displayed a lower accuracy on the training set. The smaller variance between the training and test set accuracies for the pruned DNN could again be interpreted as a sign of less overfitting.

MP-based pruning with sparsification for CNN trained on Fashion MNIST. We train the CNN found in Example 3.4 to achieve $\sim$ 92% accuracy (on the Fashion MNIST test set)

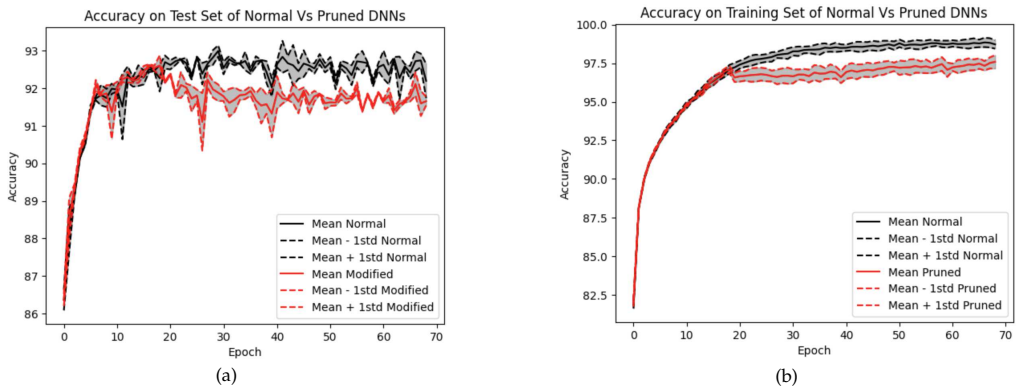
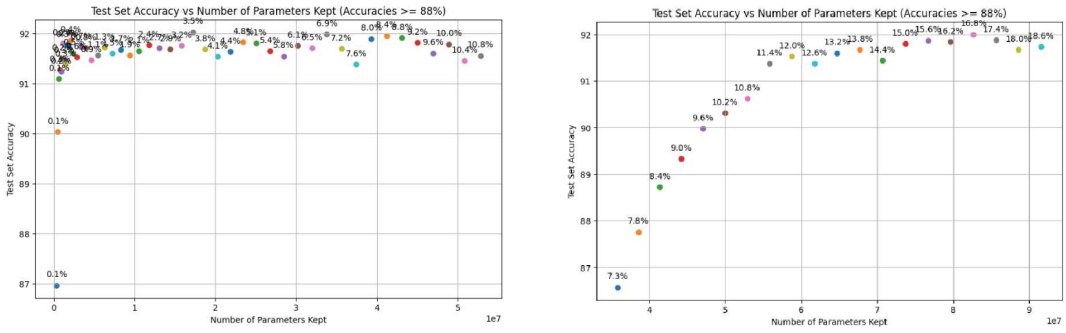


Figure 3.8: Comparison of Normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the Fashion MNIST test and training sets.

with the same MP-based pruning approach as in Section 3.2.2. At the end of the training, we employ sparsification by setting to zero all weights in the DNN smaller than some threshold ξ.

As shown in Fig. 3.9(a), this additional sparsification leads to a large reduction in parameters, by over 99.5%, without a significant drop in accuracy ($\sim$ 0.1% drop). As mentioned, Lemma 4.2 provides an explanation for why removing the small weights does not affect accuracy, as these weights appear to correspond to the noise in the weight layers, and removing them should not change accuracy.

Finally, we applied the sparsification pruning method after training without MP-based pruning (during training). Fig. 3.9(b) shows that the pruning threshold seems to affect the accuracy of the DNN in a much more significant manner. That is, even when pruning 95% of the parameters, the accuracy drops by multiple percentage points. We see that a combination of MP-based pruning with sparsification is useful for pruning while also ensuring the DNN accuracy does not decrease much for CNNs trained on Fashion MNIST.



(a) Sparsification with MP-based pruning. The percentage of parameters kept is shown above each point on the graph.

(b) Sparsification without MP-based pruning. The percentage of parameters kept is shown above each point on the graph.

Figure 3.9: Accuracy vs. number of parameters kept for CNNs trained on Fashion MNIST with and without MP-based pruning.

3.2.3 Numerics for training DNNs on CIFAR-10: Reducing parameters via MP-based pruning

In this numerical simulation, we applied the RMT algorithm to prune a DNN trained on the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 color images spanning 10 different classes. The dataset is split into a training set and a test set. The training set contains 50,000 images, while the test set comprises 10,000 images, which is standard.

Throughout the training process, we tracked the performance metrics of both the pruned and normally trained DNNs on both the test and training sets. Our analysis showed that the pruned network, despite having a reduced number of parameters, managed to achieve performance metrics comparable to those of the normally trained network. Additionally, the pruning process significantly reduced the number of parameters in the pruned DNN, resulting in a more efficient network with a lower computational footprint. The hyperparameters for the simulations in this subsection can be found in Section E.9.

The lr is also modified every epoch to be

$$lr_n = lr_{n-1} * 0.96, \tag{3.10}$$

where lr_k is the learning rate at epoch k.

Example 3.5. In this example, we trained a CNN on CIFAR10 for 350 epochs with a split frequency of 40. The convolutional layers are given by $[3, 32, 64, 128, 256, 512]$. We again apply a kernel for each layer of size 3×3.

The fully connected layers are given by $[8192, 500, 10]$. The GoF parameter for the fully connected layers is 0.08, while the GoF parameter for the convolutional layers is 0.06.

In this simulation, the portion of singular values smaller than λ_+ that we retain is given by the linear function

$$f(\text{epoch}) = \max\left(0, -\frac{1}{200} \cdot \text{epoch} + 1\right). \tag{3.11}$$

The accuracy of this DNN on the training and test set is given in Fig. 3.10.

In Example 3.2.3, it was observed that the pruned DNN exhibited a slightly lower accuracy in comparison to the normally trained DNN. Again, despite this marginal drop in performance, the pruned DNN managed to maintain this level of accuracy with much fewer parameters than was used by the normally trained DNN. While the normally trained DNN has $5,673,090$ parameters, the pruned DNN has $3,949,078$ parameters.

MP-based pruning with sparsification for CNN trained on CIFAR10. We train the CNN found in Example 3.5 on CIFAR10 to achieve $\sim$ 82% accuracy (on the test set) with the same MP-based pruning approach as in Section 3.2.3. At the end of training, we sparsify the DNN by setting to zero weights in the DNN smaller than some threshold ξ.

As shown in Fig. 3.11(a), this additional sparsification leads to a large reduction in parameters, by over 97%, without a significant drop in accuracy ($\sim$ 0 drop). Finally, we

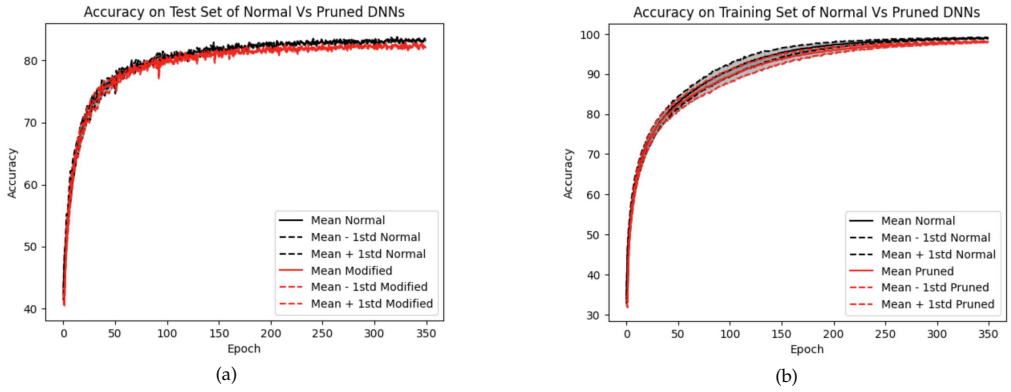
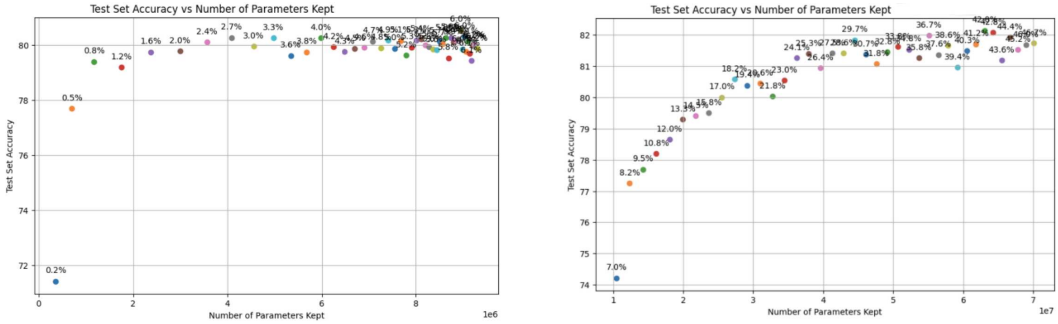


Figure 3.10: Comparison of normal DNN, trained normally, and pruned DNN, trained using the RMT approach on the test and training sets.



(a) Sparsification with MP-based pruning. The percentage of parameters kept is shown above each point on the graph.

(b) Sparsification without MP-based pruning. The percentage of parameters kept is shown above each point on the graph.

Figure 3.11: Accuracy vs. number of parameters kept for CNNs trained on CIFAR-10 with and without MP-based pruning.

tried the sparsification pruning method after training without MP-based pruning (during training). Fig. 3.11(b) shows that the threshold pruning seems to affect the accuracy of the DNN in a much more significant manner. That is, even when pruning 80% of the parameters, the accuracy drops by $\sim 2\%$. We see that a combination of MP-based pruning with sparsification is useful for pruning while also ensuring the DNN accuracy does not decrease much for CNNs trained on CIFAR-10.

4 Mathematical underpinning of numerical results

In this section, we introduce the Pruning Theorem, which provides the relationship between the accuracy of a DNN before and after being pruned. First, we introduce an important tool for this analysis, the classification confidence of a DNN.

4.1 The classification confidence

We now introduce the classification confidence, see [9]. Take $X(s,\alpha)$ to be the output of the final layer in our DNN before softmax. The classification confidence is defined as follows:

$$\delta X(s,\alpha) := X_{i(s)}(s,\alpha) - \max_{j\neq i(s)} X_j(s,\alpha). \tag{4.1}$$

In other words,

- $\delta X(s,\alpha(t)) > 0 \Rightarrow s$ is well-classified by ϕ.
- $\delta X(s,\alpha(t)) < 0 \Rightarrow s$ is misclassified by ϕ.

For T' the test set we can now define the accuracy of the DNN on T' using the classification confidence,

$$\mathrm{acc}_\alpha(t) = \frac{\#\left(\{s \in T' : \delta X(s,\alpha(t)) > 0\}\right)}{\#T'}. \tag{4.2}$$

4.2 How pruning affects classification confidence (for deterministic weight layer matrices)

Now, we state a theoretical result that shows how, at least for simple DNN models, pruning the singular values of the weight layers of a DNN impacts the DNN accuracy. This result is not based on RMT but will help in understanding the results which follow. In the following lemma, we assume that we are given a threshold $\sqrt{\lambda_+}$, which we use to prune the singular values of the layers of the DNN. In general, this threshold is given by the MP distribution and numerically can be found using the BEMA algorithm, see Section D.1. For simplicity, for W a matrix and β a bias vector, we define $(W+\beta)s := (Ws+\beta)$.

Lemma 4.1. *Let $W_1, W_2, \cdots, W_L$ and $\beta_1, \beta_2, \cdots, \beta_L$ be the weight matrices and bias vectors of a DNN with the absolute value activation function. Assume we prune a layer matrix W_b to obtain W_b' by removing singular values of W_b smaller than $\sqrt{\lambda_+}$. For any input s (either from the training set T or the test set T'), denote the change in classification confidence due to pruning as*

$$\Delta(\delta X) = \left|\delta X(s, \alpha_{W_b}) - \delta X(s, \alpha_{W_b'})\right|. \tag{4.3}$$

Here $X(s, \alpha_{W_b})$ and $X(s, \alpha_{W_b'})$ are the outputs of the final layer before softmax of the DNN with weight layer matrices W_b and W_b' respectively. Then

$$\begin{aligned}\Delta(\delta X) \leq \sqrt{2\lambda_+}\left\|\lambda \circ (W_{b-1}+\beta_{b-1}) \circ \cdots \circ \lambda \circ (W_1+\beta_1)s\right\|_2 \\ \times \sigma_{\max}(W_{b+1}) \ldots \sigma_{\max}(W_L),\end{aligned} \tag{4.4}$$

See Section E.1 for a proof of this lemma.

Remark 4.1. For the simplified case when the bias vectors are zero, this lemma says that the change in classification confidence δX after pruning is bounded by

$$\Delta(\delta X) \leq \sqrt{2\lambda_+}\|\lambda \circ W_{b-1} \circ \cdots \circ \lambda \circ W_1 s\|_2 \sigma_{\max}(W_{b+1}) \ldots \sigma_{\max}(W_L). \tag{4.5}$$

This means that if elements were well classified before the pruning and $\sqrt{2\lambda_+}\|\lambda \circ W_{b-1} \circ \cdots \circ \lambda \circ W_1 s\|_2 \sigma_{\max}(W_{b+1}) \ldots \sigma_{\max}(W_L)$ is small relative to $\delta X(s, \alpha_W)$, then after pruning s will stay accurately classified.

A crucial observation from the lemma is the product of the maximum singular values, denoted as $\sigma_{\max}(W_{b+1}) \ldots \sigma_{\max}(W_L)$. These singular values can be considerably large, implying that their product can amplify the magnitude of the bound (4.5), thereby making it substantial.

At first glance, this might seem concerning as it suggests that pruning might lead to a large drop in the network's accuracy. However, this is not necessarily a grave issue. Subsequent sections will introduce two more theoretical results based on RMT, which will elucidate why, in practice, this potential drop in classification confidence due to pruning does not occur.

Furthermore, it is crucial to note that the lemma provides a worst-case scenario. In real-world scenarios, the actual impacts of pruning are expected to be much milder than what the lemma indicates. This is a common theme in theoretical computer science and machine learning: the worst case does not always reflect the average or common case.

Furthermore, the product $\sigma_{\max}(W_{b+1})\dots\sigma_{\max}(W_L)$ is used as a naive bound on the Lipschitz constant of the function $W_L \circ \lambda \circ \cdots \circ \lambda \circ W_{b+1}$. In practice, this value can be substantially smaller. There exist other methodologies for estimating the Lipschitz constant of this function which might yield a more conservative estimate. See [18] for more on the numerical estimation of the Lipschitz constant in deep learning.

Another practical takeaway from this theorem is the preference to prune the final layers of the DNN rather than the earlier layers. The reasoning is simple: pruning the latter stages has a lesser effect on the overall accuracy, making it a safer bet in terms of maintaining the network's performance. However, this also depends on $\|\lambda \circ (W_{b-1} + \beta_{b-1}) \circ \cdots \circ \lambda \circ (W_1 + \beta_1)s\|_2$ which depends on the earlier layers in the network.

In essence, while the theorem paints a potentially alarming picture of pruning's effects, practical simulations, and further theoretical results can assuage these concerns. The nuanced understanding provided by the theorem can guide efficient pruning strategies, ensuring minimal loss in accuracy.

Example 4.1. The following example shows the histogram of δX of a trained DNN for the problem given in Section E.4. We train a DNN with one hidden layer. The weight layer matrices W_1, W_2 were initialized with components taken from i.i.d., normally distribution with zero mean and variance $1/N_l$. We obtained a 98% accuracy on the training set, which had 1000 objects. δX of the test set, after the 600th epoch of training, is given in Fig. 4.1.

For the most part, we have $\|s\|_2 \leq \sqrt{2}$ and if we were to prune the first layer of the DNN, we would obtain $\sqrt{2\lambda_+}\sigma_{\max}(W_2)\|s\|_2 \leq 6.5$. However we see that for many objects $s, \delta X$ can be much larger than 6.5.

Next we would like to obtain a better result than Lemma 4.1 using the properties of random matrices.

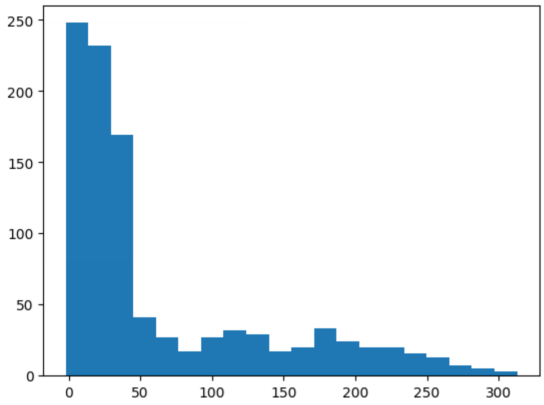


Figure 4.1: Histogram of δX of the test set for the final epoch of training. On the x-axis, we have the size of δX.

4.3 Assumptions on the random matrix R and the deterministic matrix S

We considered a class of admissible matrices W, where $W = R + S$ and W, R and S satisfy the following three assumptions. The first assumption is a condition on R.

Assumption 4.1. Assume R is a random $N \times M$ matrix with entries taken from i.i.d. with zero mean and variance $1/N$. Further, as $N \to \infty$, we have that $\sigma_{\max}(R) \to \sqrt{\lambda_+}$ a.s.

We then assume the following for the deterministic matrix S.

Assumption 4.2. Assume S is a deterministic matrix with $S = \sum_{i=1}^{r} \sigma_i u_i v_i^\top = U\Sigma V^\top$, with σ_i the singular values and u_i, $v_i^\top$ column and row vectors of U and V. Thus, S has r non-zero singular values corresponding to the diagonal entries of Σ, and all other singular values of S are zero. We also assume that these r singular values of S have multiplicity 1.

Finally, we assume for $W := R + S$.

Assumption 4.3. Take σ_i to be the singular values of S, with corresponding left and right singular vectors u_i and $v_i^\top$ and σ_i' to be the singular values of $W = R + S$, with corresponding left and right singular vectors u_i' and $v_i'^\top$. First, we assume that $N/M \to c \in (0, +\infty)$ as $N \to \infty$. Second, assume also that we know explicit functions $g_{\sigma_i,R}$, $g_{v_i,R}$ and $g_{u_i,R}$ such that as $N \to \infty$:

$$\sigma_i'(W) \xrightarrow{a.s.} \begin{cases} g_{\sigma_i,R}, & \sigma_i > \bar{\theta}(\lambda_+), \\ \sqrt{\lambda_+}, & \sigma_i < \bar{\theta}(\lambda_+), \end{cases} \tag{4.6}$$

$$|\langle u_i', u_i \rangle|^2 \xrightarrow{a.s.} \begin{cases} g_{u_i,R}, & \sigma_i > \bar{\theta}(\lambda_+), \\ 0, & \sigma_i < \bar{\theta}(\lambda_+), \end{cases} \tag{4.7}$$

$$|\langle v_i', v_i \rangle|^2 \xrightarrow{a.s.} \begin{cases} g_{v_i,R}, & \sigma_i > \bar{\theta}(\lambda_+), \\ 0, & \sigma_i < \bar{\theta}(\lambda_+). \end{cases} \tag{4.8}$$

Third, also assume that for $i \neq j$:

$$|\langle v_i', v_j \rangle|^2 \xrightarrow{a.s.} 0, \tag{4.9}$$

$$|\langle u_i', u_j \rangle|^2 \xrightarrow{a.s.} 0. \tag{4.10}$$

Here we take $\bar{\theta}(\lambda_+)$ to be a known explicit function depending on λ_+, for example see (A.2). In the Pruning Theorem 4.1, we assume that a weight layer W_b of the DNN satisfies Assumptions 4.1-4.3, that is $W_b = R_b + S_b$, with R_b a random matrix, S_b a low-rank deterministic matrix, and that the non-zero singular values of S_b are bigger than some threshold $\bar{\theta}(\lambda_+)$. Empirically, it has been observed that these assumptions are reasonable for weight matrices of a DNN, see [54, 57]. There are various spiked models in which Assumption 4.3 holds, for more on the subject see [2,4–6,8,12,15,17,17,32,41,42,65]. Also, a number of works in RMT addressed the connection between a random matrix R and the singular values and singular vectors of the deformed matrix $W = R + S$, see [7, 8]. For example, one can show that the following two simpler assumptions on the matrices R and S are sufficient to ensure that R and S satisfy the above Assumptions 4.1-4.3. Recall that a bi-unitary invariant random matrix R is a matrix with components taken from i.i.d. such that for any two unitary matrices U and $V^\top$, the components of the matrix $URV^\top$ have the same distribution as the components of R. We then assume:

Assumption 4.4 (Statistical Isotropy)**.** Assume R to be a bi-unitary invariant random $N \times M$ matrix with components taken from i.i.d. with zero mean and variance $1/N$.

We then assume the following for the deterministic matrix S:

Assumption 4.5 (Low Rank of Deterministic Matrix)**.** Assume S is a deterministic matrix with $S = \sum_{i=1}^{r} \sigma_i u_i v_i^\top = U\Sigma V^\top$, with σ_i the singular values and $u_i, v_i^\top$ column and row vectors of U and V. Thus, S has r non-zero singular values contained on the diagonal entries of Σ, and all other singular values are zero. We also assume that these r singular values of S have multiplicity 1. Finally, we assume that $N/M \to c \in (0, +\infty)$ as $N \to \infty$.

An explicit relationship between Assumptions 4.4-4.5 and Assumptions 4.1-4.3 can be found in [8]. The Assumption 4.4 is indeed strong, as it implies that the random matrix R is random in every direction. In other words, for any unitary matrices U and $V^\top$, the matrix $URV^\top$ has the same distribution as R. Random matrices with complex Gaussian entries, also known as Ginibre matrices, are a class of random matrices that are bi-unitary invariant [29].

Assumptions 4.2 and 4.5 are related to the low-rank property of the deterministic matrix, see [54, 57] for how this assumption is related to DNNs. We consider the case where we initialize the weight layer of a DNN using a Gaussian random matrix (see Example A.1) and, after training, we obtain that $W_l = R_l + S_l$ with R_l still a Gaussian random matrix and S_l having low rank. The Pruning Theorem 4.1 can be then employed to determine that removing the small singular values of W_l will not affect much the accuracy of the DNN. This is because the deformed model $W_l = R_l + S_l$ satisfies Assumptions 4.1-4.3, see Example A.1. This insight can be used to reduce the number of parameters in the DNN without sacrificing its performance, as will be further discussed in Section D.3.

We now formulate theoretical results that provide a rigorous relationship between pruning and accuracy. Note that these results are applicable to DNNs with the following architecture: Consider a DNN, denoted by ϕ, with weight layer matrices $W_1, \cdots, W_n$ and the absolute values activation function. We assume the layer maps of the DNN are compositions of linear maps and activation functions; however, the results can also be adapted to the case when the DNN is a composition of affine maps composed with activation functions, that is when we add basis. The central idea in these results can be described as follows. Suppose a weight layer W_l of the DNN satisfies the above Assumptions 4.1-4.3. Then, the removal of small singular values of W_l, smaller than the MP-based threshold $\sqrt{\lambda_+}$, does not change the classification confidence of an object $s \in \mathbb{R}^n$ by a "large amount" (see (4.14) and (4.18)). That is, the classification confidence before pruning and after pruning are essentially the same for sufficiently large matrix W_l.

In essence, these results suggest that it is possible to maintain the performance of the DNN while reducing the number of parameters by eliminating the small singular values, which are considered less influential in terms of the network's overall accuracy. This insight can be used to create more efficient DNN architectures, leading to reduced computational complexity and memory requirements without sacrificing model performance, see Appendix C.

4.4 Key technical lemma: Removing random weights for DNN with arbitrary many layers does not affect classification confidence

First, we introduce a result based on the assumption that we can directly know what parts of the weight layer matrices are deterministic and what parts are random. For a DNN ϕ with weight layer matrices $W_1, \cdots, W_L$ we start by defining,

$$g_\phi(s,b) := \|\lambda \circ W_{b-1} \circ \cdots \circ \lambda \circ W_1 s\|_2 \sigma_{\max}(W_{b+1}) \dots \sigma_{\max}(W_L), \tag{4.11}$$

$$h_\phi(s,b) := \|\lambda \circ W_{b-1} \circ \cdots \circ \lambda \circ W_1 s\|_1 \|W_{b+1}\|_1 \dots \|W_L\|_1, \tag{4.12}$$

where s is an element of the test or training set. The ℓ_1 norm of a matrix W, denoted as $\|W\|_1$, is defined as the maximum absolute column sum of the matrix. Formally, if W is an $m \times n$ matrix with entries w_{ij}, then

$$\|W\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |w_{ij}|.$$

Recall that $\sigma_{\max}(W_{b+1}) \dots \sigma_{\max}(W_L)$ is a theoretical bound on the Lipschitz constant of $W_{b+1} \cdots W_L$, but numerically one might be able to obtain a better bound, see e.g. [18]. Here, we call the Lipschitz constant of a matrix A the Lipschitz constant of the linear map corresponding to that matrix, and this definition extends to the product of matrices.

The following lemma describes how the classification confidence changes when the weight layer matrix $W_b = R_b + S_b$ is changed with the weight layer matrix S_b – the ultimate pruning.

Lemma 4.2. *Take ϕ to be a DNN with weight layer matrices $W_1, \cdots, W_L$ and absolute value activation function and fix object s from the test set T'. Assume for some b that $W_b = R_b + S_b$, with R_b a $N \times M$ random matrix satisfying Assumption 4.1 and matrix S_b a deterministic matrix satisfying Assumption 4.2.*

Suppose we replace the weight layer matrix W_b with the deterministic matrix S_b. Then we have that there exists $D(N), a(N), b(N)$ such that for the classification confidence threshold of the non-pruned DNN

$$E := a(N) h_\phi(s,b) + b(N), \tag{4.13}$$

we have the conditional probability

$$\mathbb{P}\big(\delta X(s, \alpha_{S_b}) \ge 0 \mid \delta X(s, \alpha_{W_b}) \ge E\big) \ge 1 - D(N) \tag{4.14}$$

with $D(N), a(N), b(N) \to 0$ as $N \to \infty$ and $h_\phi(s,b)$ coming from (4.12). Here, α_{S_b} are the parameters of the DNN, which has the weight matrix S_b and α_{W_b} are the parameters of the DNN with the weight layer matrix W_b.

Here, when R_b has components i.i.d. from $N(0, 1/N)$ then

$$a(N) = \frac{2}{N^{1.5/4}} + \sqrt{\frac{2 \log N^2}{N}}, \quad b(N) = 0,$$

and we have that

$$D(N) = 2\exp\left(-\frac{N^{1/4}}{2}\right).$$

The proof for this lemma can be found in Section C.1.

Remark 4.2. Here, we take s from the test set T'; however, the result also holds if we take s from the training set T. Furthermore, using a proof similar to the one given in Section C.1, one can show a more general result. That is, taking

$$\Delta(\delta X) := |\delta X(s, \alpha_{S_b}) - \delta X(s, \alpha_{W_b})|, \tag{4.15}$$

we have that

$$\mathbb{P}\big(\Delta(\delta X) \leq E\big) \geq 1 - D(N). \tag{4.16}$$

Remark 4.3. Lemma 4.2 addresses the removal of parameters while preserving accuracy in a more general context than MP-based pruning. In particular, it also explains the numerics of parameter removal via sparsification (see Section 3). Indeed, in Lemma 4.2, the random matrix R_b has entries taken from i.i.d. with zero mean and variance $1/N$. Therefore, as $N \to \infty$ the entries of R_b are small with respect to sparsification threshold $\xi(N)$. If, in addition, we assume that the entries of S_b are large (c.f. assumption in Theorem 4.1), then large entries of $W_b = R_b + S_b$ are entries of S_b with high probability. Therefore, sparsifying W_b by removing the entries smaller than the $\xi(N)$ amounts to replacing the weight layer matrix W_b with the deterministic matrix S_b. Therefore, Lemma 4.2 implies that sparsification preserves accuracy in the sense of (4.14).

Remark 4.4. Imagine you are trying to predict the weather. Initially, your prediction is based on both the randomized patterns you have observed over time (noise in input layer) and your initial random weights of the DNN (matrix R) and the deterministic factors you are sure of (matrix S). Now, if you decide to base your prediction just on the deterministic factors (that is, totally remove the random part of the weight layer matrix). Then how much would your confidence in the prediction change? This lemma provides a bound on that change.

The lemma states that there exists a function $D(N) \to 0$ as $N \to \infty$ (the size of the matrix) increases. The magnitude of this change in classification confidence (how much our "confidence" drops when we remove the random part) is given by E, which is related to the combined effects of all layers up to b and the maximum scaling factors (or singular values) of layers after b.

Most importantly, the conditional probability states that if our initial confidence (with the random matrix) was bounded away by $E > 0$, then after removing the randomness, our confidence would most likely be at least 0. And as the size N of the matrix increases, the probability that our classification confidence would be bigger than 0 becomes closer to 1.

In essence, even if we remove the randomness from our prediction model (in this case, the DNN), we can still be quite confident about our predictions, especially as our layer widths grow.

Remark 4.5. We want to understand what happens to the classification confidence threshold E in (4.14) as $N \to \infty$. Assuming that

$$h_\phi(s,b) \leq C \tag{4.17}$$

for all N, then $E \to 0$ as $N \to \infty$. This is because

$$a(N) = \frac{2}{N^{1.5/4}} + \sqrt{\frac{2\log N^2}{N}},$$

goes to zero as N increases. Consequently, the contribution from $a(N)$ becomes negligible, implying that the effect of dropping the random matrix and only keeping the deterministic matrix becomes inconsequential.

4.5 Pruning Theorem for DNN with arbitrary many layers: How pruning random weights using PM distribution affects the classification confidence

Theorem 4.1 (The Pruning Theorem for a Single Object)**.** *Take ϕ to be a DNN with weight layer matrices $W_1, \cdots, W_n$ and absolute value activation functions and take some $s \in T'$, with T' the test set. Assume for some b that $W_b = R_b + S_b$, with R_b a $N \times M$ random matrix satisfying Assumption 4.1, matrix S_b a deterministic matrix satisfying Assumption 4.2, and $R_b + S_b$ satisfying Assumption 4.3. Further, assume that all the non-zero singular values of S are bigger than $\bar{\theta}(\lambda_+)$ with λ_+ given by the MP distribution of the ESD of $R_b^\top R_b$ as $N \to \infty$ and $\bar{\theta}(\lambda_+)$ given in (4.6).*

Construct the truncated matrix W_b' by pruning the singular values of W_b smaller than $\sqrt{\lambda_+} + \epsilon$ for any ϵ. Then we have that there exists an explicit function $f_{W_b} > 0$ such that for any ϵ, there exists $C_\epsilon(N)$ so that for the classification confidence threshold of the non-pruned DNN

$$E' := (1+\epsilon)\left(\sqrt{2}(1+\epsilon)\min\left\{f_{W_b}, \sqrt{\lambda_+}\right\} g_\phi(s,b) + a(N)h_\phi(s,b) + b(N)\right), \tag{4.18}$$

we have the conditional probability

$$\mathbb{P}\left(\delta X(s, \alpha_{W_b'}) \geq 0 \mid \delta X(s, \alpha_{W_b}) \geq E'\right) \geq 1 - C_\epsilon(N) \tag{4.19}$$

with $C_\epsilon(N), a(N), b(N) \to 0$ as $N \to \infty$, $g_\phi(s,b)$ coming from (4.11) and $h_\phi(s,b)$ coming from (4.12). f_{W_b} is given in Lemma B.2. Also, α_{W_b} are the parameters of the DNN that has the weight matrix W_b and $\alpha_{W_b'}$ are the parameters of the DNN with the weight layer matrix W_b'.

See Section C.1 for a proof of this theorem.

Remark 4.6. Here we take s from the test set T'; however, the result also holds if we take s from the training set T. Furthermore, using the proof given in Section C.1, one can show a more general result. That is, taking

$$\Delta(\delta X) := \left|\delta X(s, \alpha_{W_b'}) - \delta X(s, \alpha_{W_b})\right|, \tag{4.20}$$

we have that

$$\mathbb{P}\left(\Delta(\delta X) \leq E'\right) \geq 1 - C_\epsilon(N). \tag{4.21}$$

Remark 4.7. The truncation of the matrix W_b is done by pruning the singular values smaller than $\sqrt{\lambda_+} + \epsilon$. This choice is made because, as $N \to \infty$, the largest singular value $\sigma_{\max}(R_b)$ of the random matrix R_b converges to $\sqrt{\lambda_+}$ a.s. However, for finite N, there can be fluctuations around λ_+ due to the inherent randomness of the matrix. These fluctuations are described by the Tracy-Widom distribution. To account for these fluctuations and ensure robustness in the pruning process, we add a small positive ϵ to $\sqrt{\lambda_+}$.

In the numerical part of the paper, the value of λ_+ was determined using the BEMA algorithm (see Section D.1.) This algorithm approximates λ_+ by incorporating the Tracy-Widom distribution to account for the finite-size effects and the fluctuations of the largest eigenvalue of $R_b^\top R_b$. By using this algorithm, we obtain a more accurate estimation of λ_+ for practical, finite-dimensional settings, which is crucial for effectively applying the pruning theorem in real-world scenarios.

Pruning Theorem 4.1 shows that if we replace the matrix W_b with a truncated matrix W_b', then for any given object $s \in T'$, we have that if the classification confidence $\delta X(s, \alpha)$ is positive enough for matrix W_b, it stays positive for the truncated matrix W_b' with high probability. In other words, almost all well-classified objects remain well-classified after replacing W_b with W_b'. We also show numerically that it is easier to prevent overfitting using matrix W_b' instead of the larger matrix W_b. We verified that removing small singular values based on the MP-based threshold $\sqrt{\lambda_+}$ for the case when the weight matrices W_b were initialized with $N(0, 1/N)$ does not reduce the accuracy of the DNN, see Example D.3. Here, $f_W = \|W' - S\|$ and for a large class of RMT matrix models (see Assumptions 4.1-4.3), for the case $N \to \infty$, we obtain f_W based on the singular values of W only.

Remark 4.8. The assumptions made in Theorem 4.1 are quite natural and hold for a wide range of DNN architectures. Assumption 4.1 focuses on the random matrix R. This assumption ensures that the random matrix R captures the essential randomness in the weight layer while also satisfying the requirements given in Theorem 3.1.

Assumption 4.2 pertains to the deterministic matrix S, which is assumed to have a specific structure, with r non-zero singular values and all other singular values being zero. Moreover, these r singular values have multiplicity 1, which is a reasonable expectation for a deterministic matrix that contributes to the information content in the layer W_b. Assumption 4.3 holds for many spiked models and has been studied in much detail.

The assumption that the singular values of the deterministic matrix S are larger than some $\bar{\theta}(\lambda_+)$ is also quite natural, see [54, 57]. This is because the deterministic matrix S represents the information contained in the weight layer, and its singular values are expected to be large, reflecting the importance of these components in the overall performance of the DNN. On the other hand, the random matrix R captures the inherent randomness in the weight layer, and with high probability depending on N, its singular values should be smaller than the MP-based threshold. This means that there is a clear boundary between the information and noise in the layer W_b, which is also natural, see [54].

This distinction between the singular values of S and R highlights the separation between the information and noise in the weight layer, allowing us to effectively remove the small singular values without impacting the accuracy of the DNN. The assumption thus

provides a solid basis for studying the behavior of DNNs with weight layers modeled as spiked models. It contributes to our understanding of the effects of removing small singular values based on the random matrix theory MP-based threshold $\sqrt{\lambda_+}$.

Remark 4.9. In our work, we leverage the Marchenko-Pastur distribution to select significant singular values for the low-rank approximation of our weight layers W_l. Other low-rank approximation techniques, such as the bootstrapping technique proposed in [40], could potentially be integrated with the Marchenko-Pastur distribution to further refine the low-rank approximation of W_l.

4.6 Simple example of DNN with one hidden layer

The following is a simple example of the Pruning Theorem:

Example 4.2. Take ϕ to be a DNN with three weight layer matrices W_1, W_2, W_3 and the absolute value activation function and take $s \in T'$. This is

$$\phi(s, \alpha) = \rho \circ \lambda \circ W_3 \circ \lambda \circ W_2 \circ \lambda \circ W_1 s, \quad s \in \mathbb{R}^n. \tag{4.22}$$

Assume $W_2 = S_2 + R_2$ satisfies Assumptions 4.1-4.3 and W_1, W_3 are arbitrary. More specifically, assume R_2 to be a random matrix with i.d.ds taken from the distribution $N(0, 1/N)$ and S_2 to be a $N \times N$ deterministic matrix with non-zero singular values bigger than 1.

Take W' to be the same as W but with all the singular values of W smaller than $2 + \epsilon$, for any ϵ, set to zero. Then for f_W the positive function given in (4.24) we have that for any $\epsilon > 0$,

$$\begin{aligned} &\mathbb{P}\Big(\delta X(s, \alpha_{W'}) \geq 0 \mid \delta X(s, \alpha_W) \geq (1+\epsilon)\sqrt{2} \\ &\qquad \times \big(a(N)\|W_1 s\|_1 \|W_3\|_1 + f_{W_2}\|W_1 s\|_2 \sigma_{\max}(W_3) + b(N)\big)\Big) \\ &\geq 1 - C_\epsilon(N) \end{aligned} \tag{4.23}$$

with $C_\epsilon(N), a(N), b(N) \to 0$ as $N \to \infty$.

Here we have that for σ_r the smallest non-zero singular value of S_2

$$f_W = \max_{1 \leq i \leq r} \left\{ \sqrt{\left(\sigma_i^2 + \left(\frac{1+\sigma_i^2}{\sigma_i}\right)^2 \left(1 - \frac{1}{\sigma_i^2}\right)\right) - 2(1+\sigma_i^2)\left(1 - \frac{1}{\sigma_i^2}\right)} \right\}. \tag{4.24}$$

Note that as $\sigma_r \to \infty$ we have $f_W \to 1$. In fact, we show numerically that already for $\sigma_r \geq 5$ we have $|f_W - 1| \leq .03$, see Fig. 4.2.

Note that this estimate is given in terms of the singular values of S (which are σ_i). However, the singular values of S might not be known. Nevertheless, by Theorem A.1, we have that as $N \to \infty$ the singular values of S can be obtained directly from the singular values of W via $\sigma_i' = (1+\sigma_i^2)/\sigma_1$. Thus, as $N \to \infty$ this estimate can be obtained in terms of singular values of W only, which is why we use the notation f_W and not f_S. For simplicity, we keep using the current notation.

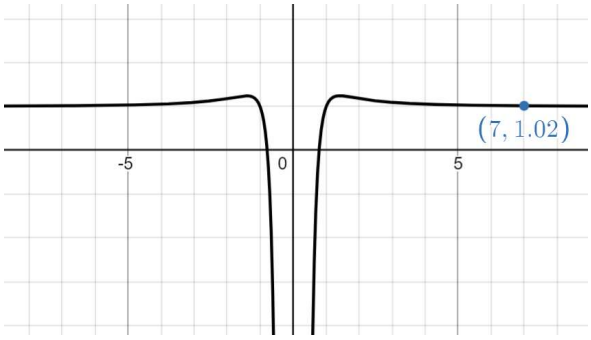


Figure 4.2: Graph of f_W. On the x-axis, we have σ_i.

We numerically checked that for R a 3000×3000 random matrix initialized with the above Gaussian distribution, and for S a diagonal matrix with 5 non-zero singular values given by $30, 40, 50, 60, 70$, we have $\|S - W'\|_2 \approx f_W \approx 1$, see Fig. 4.3. Thus, in this example $f_W < \sqrt{\lambda_+}$, given that $\sqrt{\lambda_+} = 2$, and so in (C.22) we would have $\min\{f_{W_b}, \sqrt{\lambda_+}\} \approx 1$. Thus, Theorem 4.1 provides a better result than Lemma 4.1. For more on this example, see Section B.1.

It is an important question: Under what conditions of R and S would we have that $f_W = c$ such that $c < \sqrt{\lambda_+}$.

Example 4.3. Consider R to be an $n \times n$ symmetric (or Hermitian) matrix with independent, zero mean, normally distributed entries. The variance of the entries is σ^2/n on the diagonal and $\sigma^2/(2n)$ on the off-diagonal.

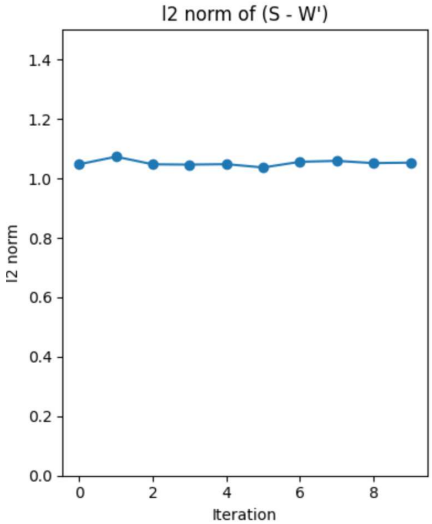


Figure 4.3: The norm $\|S - W'\|_2$ is shown for 10 random matrices with elements i.i.d. taken from $N(0, 1/N)$ and S a diagonal matrix with elements on the diagonal given by $30, 40, 50, 60, 70$. We see that the norm is very close to $f_W \approx 1$.

In the setting where $S = \sum_{i=1}^{r} \sigma_i u_i u_i^\top$, let u_i' be the unit eigenvectors of $W = R + S$ associated with its r largest eigenvalues. Assuming that for all $1 \leq i \leq r$ we have $\sigma_i > \sigma$, then

$$f_W = \max_{1 \leq i \leq r} \sqrt{\left(\sigma_i^2 + \left(1 - \frac{\sigma^2}{\sigma_i^2}\right)\left(\sigma_i + \frac{\sigma^2}{\sigma_i}\right)^2\right) - 2\sigma_i\left(\left(\sigma_i + \frac{\sigma^2}{\sigma_i}\right)\left(1 - \frac{\sigma^2}{\sigma_i^2}\right)\right)}. \tag{4.25}$$

In this example, $\sqrt{\lambda_+} = 2\sigma$. Thus, in Fig. 4.4 we compare f_W vs 2σ. As mentioned, it would be interesting to try and find a probability distribution which, if R is initialized with would result in a very small f_W for reasonable assumptions on the singular values of S.

See Section B.2 for more information.

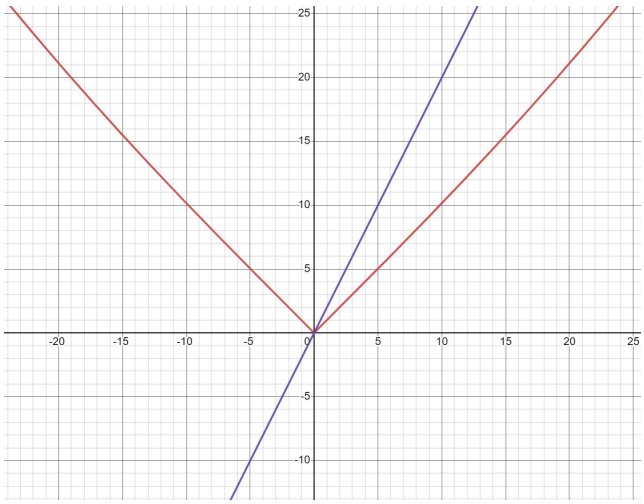


Figure 4.4: A graphical comparison between f_W and 2σ as functions of σ. The x-axis represents the variable σ ranging from -25 to 25. The y-axis provides the computed values for both f_W and 2σ. We see that $f_W \approx |\sigma|$, which is smaller than 2σ for positive σ. Note, to obtain this numerical result, we must ensure that $\sigma_r \geq \sigma$.

4.7 Pruning Theorem for accuracy: How pruning affects accuracy

In this subsection, we present a version of the Pruning Theorem for accuracy, which describes how the accuracy of a DNN is affected by pruning. We present this theorem for DNNs with one hidden layer. However, it can be generalized for DNNs with more layers.

We first recall the notion of the good set of a DNN, introduced in [9]. The good set is a subset of the test set T' defined as follows: For $\eta \geq 0$, the good set of margin η at time t is

$$G_{\eta(t),\alpha} := \{s \in T' : \delta X(s, \alpha(t)) > \eta\}. \tag{4.26}$$

Basically, the good set consists of positively classified objects whose classification confidence is bounded below by η. Next, we formulate the Pruning Theorem for accuracy.

Loosely speaking, it says that for some threshold E_{acc} (see (4.29)), we have that the accuracy of the DNN after pruning is bounded from below by the number

$$\frac{|G_{E_{acc},\alpha}|}{|T'|}, \tag{4.27}$$

where for a finite set A, we have $|A|$, which is the number of elements in that set.

Theorem 4.2 (Pruning Theorem for Accuracy)**.** *Let ϕ be a DNN with weight layer matrices W_1, W_2, W_3, and λ the absolute value activation function*

$$\phi(s,\alpha) = \lambda \circ W_3 \circ \lambda \circ W_2 \circ \lambda \circ W_1 s, \quad s \in \mathbb{R}^n. \tag{4.28}$$

Assume $W_2 = S_2 + R_2$ satisfies Assumptions 4.1-4.3, and W_1, W_3 are arbitrary matrices.

Construct the truncated matrix W_2' by pruning singular values of W_2 smaller than $\sqrt{\lambda_+} + \epsilon$, for any ϵ. For every ϵ, introduce the classification confidence threshold for the non-pruned DNN as the smallest number $E_{acc} \geq 0$ for which we satisfy

$$\begin{aligned} E_{acc} = (1+\epsilon)\Big(\sqrt{2}\big(f_{W_2}\sigma_{\max}(W_3) + a(N)\|W_3\|_1\big)\Big) \\ \times \max_{s \in G_{E_{acc},\alpha}} \|W_1 s\|_1 + b(N) \quad (\textit{positive}) \end{aligned} \tag{4.29}$$

with $f_{W_2} > 0$ an explicit rational function of W_2. Then we have for any ϵ,

$$\mathbb{P}\left(acc_{\alpha'}(t) \geq \frac{|G_{E_{acc},\alpha}|}{|T'|}\right) \geq \big(1 - C_\epsilon(N)\big)^{|G_{E_{acc}\alpha}|} \tag{4.30}$$

with $C_\epsilon(N), a(N), b(N) \to 0$ as $N \to \infty$. Here α, α' are the parameters of the non-pruned and pruned DNNs, respectively and $acc_{\alpha'}(t)$ is given in (4.2).

See Section C.2 for a proof of this theorem.

Remark 4.10. Theorem 4.2 applies to the entire training and test set. It is important to note that if the sizes of the training and test sets depend on the matrix dimension N, then the properties of the good set, which is derived from these sets, will inherently depend on N as well. The specifics of this dependency remain undefined within the current scope of our analysis.

Remark 4.11. One can always find a $E_{acc} \geq 0$ which satisfies (4.29). This is because

$$0 \leq (1+\epsilon)\Big(\sqrt{2}\big(f_{W_2}\sigma_{\max}(W_3) + a(N)\|W_3\|_1\big)\Big) \max_{s \in G_{0,\alpha}} \|W_1 s\|_1.$$

Finally, for large enough η, we have

$$(1+\epsilon)\Big(\sqrt{2}\big(f_{W_2}\sigma_{\max}(W_3) + a(N)\|W_3\|_1\big)\Big) \max_{s \in G_{\eta,\alpha}} \|W_1 s\|_1 = 0.$$

Appendix A Some known results on perturbation of matrices

Matrix perturbation theory is concerned with understanding how small changes in a matrix can affect its properties, such as eigenvalues, eigenvectors, and singular values. In this section, we state a couple of known results from matrix perturbation theory.

A.1 Asymptotics of singular values and singular vectors of deformation matrix

The results in this subsection are taken from [8]. Given the Assumptions 4.4-4.5 on R and S described in Section 4.3, the authors were able to show that the largest eigenvalues and corresponding eigenvectors of $W = S + R$ are well approximated by the largest eigenvalues and eigenvectors of S.

We start by defining the following function:

$$D_{\mu_R}(z) = \left[\int \frac{z}{z^2 - t^2} d\mu_R(t)\right] \times \left[c \int \frac{z}{z^2 - t^2} d\mu_R(t) + \frac{1-c}{z}\right] \tag{A.1}$$

for $z > \sqrt{\lambda_+}$, with λ_+ given by the MP distribution of $R^\top R$. Take $D^{-1}_{\mu_R}(\cdot)$ to be its functional inverse. Set

$$\bar{\theta} = D_{\mu_R}(\sqrt{\lambda_+})^{-\frac{1}{2}}. \tag{A.2}$$

Theorem A.1 (Theorem for Large Singular Values, [8]). *Take $W = R + S$, with W, R and S all $N \times M$ matrices satisfying Assumptions 4.4-4.5. The r largest singular values of W, denoted as $\sigma'_i(W)$ for $1 \le i \le r$, exhibit the following behaviour as $N \to \infty$:*

$$\sigma'_i(W) \xrightarrow{a.s.} \begin{cases} D^{-1}_{\mu_R}\left(\frac{1}{(\sigma_i)^2}\right), & \sigma_i > \bar{\theta}, \\ \sqrt{\lambda_+}, & \sigma_i < \bar{\theta}. \end{cases} \tag{A.3}$$

Theorem A.2 (Norm of Projection of Largest Singular Vectors, [8]). *Take indices $i_0 \in \{1, \dots, r\}$ such that $\sigma_{i_0} > \bar{\theta}$. Take $\sigma'_{i_0} = \sigma'_{i_0}(W)$ and let u', v' be left and right unit singular vectors of W associated with the singular value σ'_{i_0} and u, v be the corresponding singular vectors of S. Then we have, as $N \to \infty$*

$$\left|\left\langle u', \mathrm{Span}\{u_i \ \ s.t. \ \ \sigma_i = \sigma_{i_0}\}\right\rangle\right|^2 \xrightarrow{a.s.} \frac{-2\phi_{\mu_R}(\rho)}{\sigma^2_{i_0} D'_{\mu_R}(\rho)}, \tag{A.4}$$

$$\left|\left\langle v', \mathrm{Span}\{v_i \ \ s.t. \ \ \sigma_i = \sigma_{i_0}\}\right\rangle\right|^2 \xrightarrow{a.s.} \frac{-2\phi_{\mu_R}(\rho)}{\sigma^2_{i_0} D'_{\tilde{\mu_R}}(\rho)}. \tag{A.5}$$

Here

$$\rho = D^{-1}_{\mu_R}\left(\frac{1}{(\sigma_{i_0})^2}\right), \quad \tilde{\mu_R} = c\mu_R + (1+c)\delta_0.$$

Further,

$$\left|\left\langle u', \mathrm{Span}\{u_i \ \ s.t. \ \ \sigma_i \neq \sigma_{i_0}\}\right\rangle\right|^2 \xrightarrow{a.s.} 0, \tag{A.6}$$

$$\left|\left\langle v', \mathrm{Span}\{v_i \ \ s.t. \ \ \sigma_i \neq \sigma_{i_0}\}\right\rangle\right|^2 \xrightarrow{a.s.} 0. \tag{A.7}$$

Example A.1. Take $S = \sum_{i=1}^{r} \sigma_i u_i v_i^\top$ to be a $N \times N$ deterministic matrix, with σ_i the singular values and v_i and u_i the singular vectors of S. Take R to be a $N \times N$ random matrix with real i.i.d. components taken from the normal distribution $N(0, 1/N)$. For $W = R + S$ we have

Theorem A.3 (Theorem for Large Singular Values for Example A.1)**.** *The r largest singular values of W, denoted $\sigma_i'(W)$ for $1 \leq i \leq r$, exhibit the following behaviour as $N \to \infty$:*

$$\sigma_i'(W) \xrightarrow{a.s.} \begin{cases} \dfrac{1+\sigma_i^2}{\sigma_i}, & \sigma_i > 1, \\ 2, & \sigma_i < 1. \end{cases}$$

Theorem A.4 (Theorem for Large Singular Vectors for Example A.1)**.** *Assuming that the r largest singular values of W have multiplicity 1, then the right and left singular vectors u_i', v_i' of W corresponding with the r largest singular values $\sigma_i'(W)$ exhibits the following behaviour as $N \to \infty$:*

$$|\langle v_i, v_i' \rangle|^2, |\langle u_i, u_i' \rangle|^2 \xrightarrow{a.s.} \begin{cases} \left(1 - \dfrac{1}{\sigma_i^2}\right), & \sigma_i > 1, \\ 0, & \sigma_i < 1. \end{cases}$$

Remark A.1. We say that $X_n \to X$ in probability if for any ϵ, $\mathbb{P}(|X_n - X| > \epsilon) \to 0$ as $n \to \infty$. One can show that $X_n \to X$ a.s. implies that $X_n \to X$ in law. Thus for Theorems A.1 and A.2 we have that if the r large singular values of S are bigger than 1 and have multiplicity 1 then there exists some constant $B_N(\epsilon)$ with $B_N(\epsilon) \to 0$ as $N \to \infty$ such that for the r largest singular values of W and their corresponding singular vectors and for any ϵ we have

$$\mathbb{P}\left(\left| \sigma_i'(W) - \frac{1+\sigma_i^2}{\sigma_i} \right| > \epsilon \right) < B_N(\epsilon), \tag{A.8}$$

$$\mathbb{P}\left(\left| |\langle v_i, v_i' \rangle|^2 - \left(1 - \frac{1}{\sigma_i^2}\right) \right| > \epsilon \right) < B_N(\epsilon), \tag{A.9}$$

$$\mathbb{P}\left(\left| |\langle u_i, u_i' \rangle|^2 - \left(1 - \frac{1}{\sigma_i^2}\right) \right| > \epsilon \right) < B_N(\epsilon). \tag{A.10}$$

Finally, we also have that

$$\mathbb{P}(|\langle u_i, u_j' \rangle| > \epsilon) < B_N(\epsilon), \tag{A.11}$$

$$\mathbb{P}(|\langle v_i, v_j' \rangle| > \epsilon) < B_N(\epsilon) \tag{A.12}$$

for $i \neq j$.

A.2 Gershgorin's circle theorem

Finally, we state Gershgorin's circle theorem.

Theorem A.5 (Gershgorin's Circle Theorem). *Let $B = [b_{ij}]$ be an $n \times n$ complex matrix. Define the Gershgorin discs D_i for $1 \leq i \leq n$ as*

$$D_i = \left\{ z \in \mathbb{C} : |z - b_{ii}| \leq \sum_{j \neq i} |b_{ij}| \right\}. \tag{A.13}$$

Then, every eigenvalue λ of the matrix B lies within at least one of the Gershgorin discs D_i.

Remark A.2. We apply Theorem A.5 for almost diagonal matrices when (A.13) estimates how close the eigenvalues are to the diagonal elements. This closeness is estimated in terms of the magnitude of the non-diagonal elements.

Appendix B An approximation lemma – pruned matrix W' approximates the deterministic matrix S

Assume we are given a deterministic matrix S and we add to it a random matrix R, for the R and S given in Example A.1. Suppose we take $W = S + R$, it is well known that one can find a rank k approximation of W. This is done by taking the SVD of $W = U\Sigma V^\top$ and setting all but the top k singular values in Σ to zero. The following is a known theorem of this result:

Theorem B.1. *Given the singular value decomposition (SVD) of $W = U\Sigma V^\top$, where U and V are unitary matrices and Σ is a diagonal matrix containing the singular values of W, the rank k approximation of W is given by*

$$\tilde{W}_k = U_k \Sigma_k V_k^\top, \tag{B.1}$$

where U_k and V_k are the matrices obtained by retaining only the first k columns of U and V, respectively, and Σ_k is the matrix obtained by retaining only the first k diagonal entries of Σ. Here we use $\tilde{W}_k$ to distinguish it from the weight layer matrix W_k. This approximation represents the best rank k approximation to W in the following sense:

$$\tilde{W}_k = \min_{rank(X)=k} \|W - X\|_F, \tag{B.2}$$

where X is an arbitrary matrix of rank k and $\|\cdot\|_F$ is the Frobenius norm.

See [22] for more on this result. In this section, we wish to obtain slightly different results in a similar direction. We wish to show that for the R and S given in Example A.1, $\tilde{W}_r$ is a good approximation of S (recall S has r non-zero singular values). That means that W_r is a good approximation of the deterministic part of W. We, therefore, state a lemma that shows that

$$\|(\tilde{W}_r - S)z\| < f_W \|z\|. \tag{B.3}$$

Rather than state this lemma in terms of $\tilde{W}_r$, we state them in terms of W', which is defined as follows.

Definition B.1. *Take $W = R + S$, with R and S given in Assumptions 4.1-4.3. Take $W = U\Sigma V^\top$ to be the SVD of W and take, for any ϵ,*

$$\Sigma'_{i,j} = \begin{cases} \Sigma_{i,j}, & \Sigma_{i,j} > \sqrt{\lambda_+} + \epsilon, \\ 0, & \Sigma_{i,j} \leq \sqrt{\lambda_+} + \epsilon. \end{cases} \tag{B.4}$$

Then we obtain the truncated matrix W' by taking $W' = U\Sigma' V^\top$.

Next, we define a padded unitary matrix, which is helpful when proving the approximation theorem.

Definition B.2 (Padded Unitary Matrix)**.** *A padded-unitary matrix Q of size $(n+m)\times(n+m)$ is defined as a matrix where the first $n \times n$ submatrix is a unitary matrix, and the remaining entries are zeros. Formally, if U is an $n \times n$ unitary matrix, then Q is constructed as*

$$Q = \begin{pmatrix} U & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix},$$

where $\mathbf{0}$ represents the appropriate-sized matrix with all zero entries.

Next, we present an approximation lemma that gives a bound for $\|(W'-S)z\|_2$, where z is any vector. This lemma describes how well S is being approximated by W'.

Approximation Lemma B.2. *Assume that $W = R + S$, with R a $N \times M$ random matrix and S a deterministic matrix satisfying Assumptions 4.1-4.3. Assume that the singular values of S are bigger than $\bar{\theta}$, given in Assumption 4.3, and all singular values have multiplicity one. Take z to be any vector in $\mathbb{R}^n$. Then for W' given in Definition B.1 we have that there exists $f_W > 0$ and exists $B^*_N(\epsilon)$ such that for any $\epsilon > 0$,*

$$\mathbb{P}\big(\|(W' - S)z\|_2 \geq (1+\epsilon) f_W \|z\|_2\big) < B^*_N(\epsilon) \tag{B.5}$$

with $B^*_N(\epsilon) \to 0$ as $N \to \infty$.

Moreover, assuming that σ_r is the smallest singular values of S, we have

$$f_W = \max_{1\leq i\leq r} \sqrt{(g_{v_i} g^2_{\sigma_i} + \sigma_i^2) - 2 g_{\sigma_i}\sigma_i \sqrt{g_{v_i}}\sqrt{g_{u_i}}} \tag{B.6}$$

with g_{σ_i}, g_{u_i} and g_{v_i} given in Assumption 4.3.

Remark B.1. Here f_W depends on the distribution of the eigenvalues of R and on the eigenvalues of S.

Example B.1. Assume that W is the matrix given in Example A.1. Then

$$f_W = \max_{1\leq i\leq r} \left\{ \sqrt{\left(\sigma_i^2 + \left(\frac{1+\sigma_i^2}{\sigma_i}\right)^2 \left(1 - \frac{1}{\sigma_i^2}\right)\right) - 2(1+\sigma_i^2)\left(1 - \frac{1}{\sigma_i^2}\right)} \right\}. \tag{B.7}$$

Example B.2. Assume that $W = R + S$, with R a $N \times M$ random matrix satisfying Assumption 4.4 and S a deterministic matrix satisfying Assumption 4.5. Assume that the singular values of S are bigger than $\bar{\theta}$, given in (A.2), and all singular values have multiplicity one. Then for W' given in Definition B.1 we have that

$$f_W = \max_{1\leq i\leq r} \sqrt{\left(\frac{-2\phi_{\mu_R}(\rho)}{\sigma_r^2 D'_{\mu_{\tilde{R}}}(\rho)} D^{-1}_{\mu_R}\left(\frac{1}{(\sigma_i)^2}\right)^2 + \sigma_i^2\right) - 2\sigma_i D^{-1}_{\mu_R}\left(\frac{1}{(\sigma_i)^2}\right)\sqrt{\frac{-2\phi_{\mu_R}(\rho)}{\sigma_r^2 D'_{\mu_{\tilde{R}}}(\rho)}}\sqrt{\frac{-2\phi_{\mu_R}(\rho)}{\sigma_r^2 D'_{\mu_R}(\rho)}}}.$$

Proof. We prove this for the simple case when $S = \sum_{i=1}^{r} \sigma_i u_i v_i^\top$ and for the Example A.1, when W, R and S are $N \times N$ matrices. The proof for the more general case is the same.

Take $W' = U_W \Sigma'_W V_W^\top$ and $S = U_S \Sigma'_S V_S^\top$ to be the SVD of W' and S, with Σ'_S a $N \times N$ matrix with r non-zero singular values σ_i on its diagonal and all other elements zero, and assume the smallest singular value of S, which is σ_r, is bigger than 1. Furthermore, take U_W, V_W to be $N \times N$ unitary matrices, and U_S, V_S to be $N \times N$ padded unitary matrices, as defined in Definition B.2, with the "singular vectors" corresponding to the zero singular values σ_i being the zero vector. If we look at the SVD in terms of its summation representation, changing the singular vectors corresponding to the zero singular values to zero vectors (which are not unit vectors) will not change the reconstructed components of the matrix S. Let us clarify this:

The SVD of a matrix S can be written as

$$S = \sum_{i=1}^{r} \sigma_i u_i v_i^\top,$$

where σ_i are the singular values, u_i are the left singular vectors, and v_i are the right singular vectors, with r being the rank of S. This sum runs over all singular values, including the zero singular values.

In this sum, each term $\sigma_i u_i v_i^\top$ contributes to the matrix S. However, for terms where $\sigma_i = 0$, the entire term $\sigma_i u_i v_i^\top$ becomes a zero matrix, regardless of what u_i and v_i are. This is because multiplying by zero annihilates any contribution from these vectors.

Therefore, if you change the singular vectors corresponding to the zero singular values (let us say, replacing them with zero vectors), these terms in the sum still contribute nothing to S because they are multiplied by zero. The non-zero singular values and their corresponding vectors still determine the matrix S, and the contributions from the terms with zero singular values remain zero.

In the adapted form of the SVD of S being considered, the matrices U and V do not strictly adhere to the traditional definition of unitary matrices. Instead, they can be described as padded unitary matrices. For the purposes of the proof in question, this "almost" unitary nature of U and V is sufficient.

Then we have

$$\begin{aligned}\|(W' - S)z\|_2 &= \left\|(U_W \Sigma'_W V_W^\top - U_S \Sigma'_S V_S^\top)z\right\| \\ &\leq \sqrt{\lambda_{\max}\left((U_W \Sigma'_W V_W^\top - U_S \Sigma'_S V_S^\top)^\top (U_W \Sigma'_W V_W^\top - U_S \Sigma'_S V_S^\top)\right)}\|z\|_2 \quad \text{(B.8)}\end{aligned}$$

with $\lambda_{\max}(A)$ the largest eigenvalue of A. Thus we obtain

$$\begin{aligned} &\|(W'-S)z\|_2 \\ &\leq \sqrt{\lambda_{\max}\big((V_W\Sigma_W'^2V_W^\top + V_S\Sigma_S'^2V_S^\top - V_W\Sigma_W'U_W^\top U_S\Sigma_S'V_S^\top - V_S\Sigma_S'U_S^\top U_W\Sigma_W'V_W^\top)\big)}\|z\|_2. \end{aligned} \tag{B.9}$$

We can multiply the right and left of the matrix

$$(V_W\Sigma_W'^2V_W^\top + V_S\Sigma_S'^2V_S^\top - V_W\Sigma_W'U_W^\top U_S\Sigma_S'V_S^\top - V_S\Sigma_S'U_S^\top U_W\Sigma_W'V_W^\top)$$

by the unitary matrices V_W and $V_W^\top$ respectively, without increasing the absolute value of the max eigenvalue, to obtain

$$\begin{aligned} &\|(W'-S)z\|_2 \\ &\leq \sqrt{\lambda_{\max}(\Sigma_W'^2 + V_W^\top V_S\Sigma_S'^2V_S^\top V_W - \Sigma_W'U_W^\top U_S\Sigma_S'V_S^\top V_W - V_W^\top V_S\Sigma_S'U_S^\top U_W\Sigma_W')}\|z\|_2. \end{aligned} \tag{B.10}$$

By (A.12), (A.11) and Theorem A.5, since the sum of the off diagonal elements of

$$G := (\Sigma_W'^2 + V_W^\top V_S\Sigma_S'^2V_S^\top V_W - \Sigma_W'U_W^\top U_S\Sigma_S'V_S^\top V_W - V_W^\top V_S\Sigma_S'U_S^\top U_W\Sigma_W') \tag{B.11}$$

can be made arbitrarily small with high probability as $N \to \infty$, there exists $B_N^*(\epsilon)$ so that for large enough N we have

$$\mathbb{P}\Big(\big|\lambda_{\max}(G) - \lambda_{\max}\big((\Sigma_W'^2 + D_2^2\Sigma_S'^2) - 2\Sigma_W'\Sigma_S'\big((D_2)D_1\big)\big)\big| > \epsilon\Big) < B_N^*(\epsilon), \tag{B.12}$$

where D_1 is the diagonal matrix containing $\langle u_i, \tilde{u}_i\rangle$ on its diagonal and all other elements zero and D_2 the diagonal matrix containing $\langle v_i, \tilde{v}_i\rangle$ on its diagonal and all other elements zero. In fact, because S only has r non-zero singular values, we obtain less than $r \times r$ non-zero off-diagonal elements for the matrices $U_W^\top U_S, U_S^\top U_W, V_W^\top V_S$, and $V_S^\top V_W$. Thus, because r is fixed we have by Theorems A.3, A.4 and Remark A.1, that there exists $B_N^*(\epsilon)$ such that for large enough N

$$\begin{aligned} &\mathbb{P}\left(\|(W'-S)z\|_2 \geq (1+\epsilon)\right. \\ &\qquad\left.\times \max_{1\leq i\leq r}\left\{\sqrt{\left(\sigma_i^2 + \left(\frac{1+\sigma_i^2}{\sigma_i}\right)^2\left(1-\frac{1}{\sigma_i^2}\right)\right) - 2(1+\sigma_i^2)\left(1-\frac{1}{\sigma_i^2}\right)}\|z\|_2\right\}\right) \\ &\leq B_N^*(\epsilon). \end{aligned} \tag{B.13}$$

In fact, using the above argument, we can show that as $N \to \infty$

$$\|(W'-S)\| \xrightarrow{a.s.} \max_{1\leq i\leq r}\left\{\sqrt{\left(\sigma_i^2 + \left(\frac{1+\sigma_i^2}{\sigma_i}\right)^2\left(1-\frac{1}{\sigma_i^2}\right)\right) - 2(1+\sigma_i^2)\left(1-\frac{1}{\sigma_i^2}\right)}\right\}. \tag{B.14}$$

This completes the proof. □

B.1 Numerics for Example 4.2

In the following subsection, we provide a figure of the dot products of the 5 left and right singular values for the matrices $W = R + S$ and S described in Example 4.2, see Fig. B.1. As mentioned earlier, the 5 singular values of S were $30, 40, 50, 60, 70$. We see that the dot product of the left and right singular vectors of W and S can be approximated almost perfectly by the equation $\sqrt{1 - 1/\sigma_i^2}$, see (A.8). That is, for $\sigma_5 = 30$, we have $\sqrt{1 - 1/\sigma_i^2} \approx 0.99944$ and indeed $\langle u_5, u_5' \rangle \approx 0.99943$ and similarly for the other dot products.

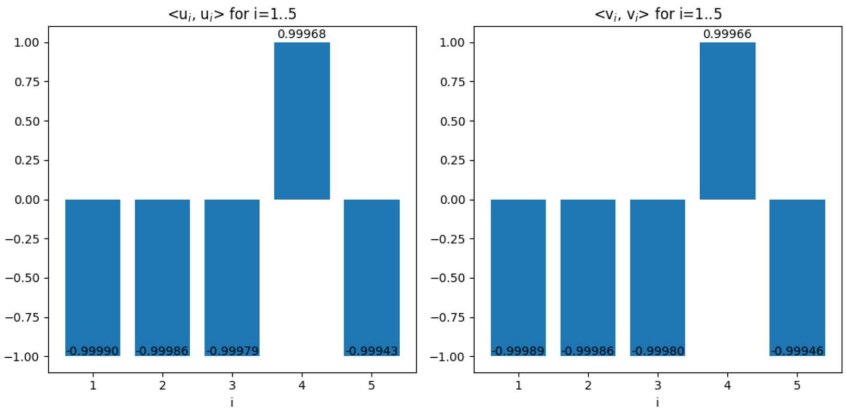


Figure B.1: Left: Dot product of the left singular vectors. Right: Dot product of the right singular vectors.

B.2 Details for Example 4.3

The details provided in the subsection were taken from [7]. Under the setting given in Example 4.3, let u_i' be a unit-norm eigenvector of $R + S$ associated with its r largest eigenvalues. We have for $1 \leq i \leq r$,

$$\lambda_i(R+S) \xrightarrow{\text{a.s.}} \begin{cases} \sigma_i + \dfrac{\sigma^2}{\sigma_i}, & \text{if} \quad \sigma_i > \sigma, \\ 2\sigma, & \text{otherwise,} \end{cases}$$

as $n \to \infty$. We also have

$$|\langle u, u' \rangle|^2 \xrightarrow{\text{a.s.}} \begin{cases} 1 - \dfrac{\sigma^2}{\sigma_i^2}, & \text{if} \quad \sigma_i > \sigma, \\ 0, & \text{otherwise.} \end{cases}$$

Appendix C Proof for Pruning Theorem

The proof of the Pruning Theorem 4.1 consists of two steps. First, we show how classification confidence changes when the weight matrix W_b is replaced with its deterministic

part S_b. Second, we approximate the deterministic matrix S_b with the pruned matrix W_b', keeping track of the corresponding change in the classification confidence. The key idea in the second step is to use asymptotics of the spectrum of deformed matrices $W = R + S$ with a finite number of singular values of S, see [8], in combination with GCT.

C.1 Proof for key technical Lemma 4.2

Proof. We first provide a proof for the case when our DNN has only one layer matrix W.

We proceed by showing that if $\|s\|_2$ is independent of N then as $N \to \infty$,

$$\delta X(s, \alpha_W) - \delta X(s, \alpha_S) = 0$$

in probability, with α_W the parameters of the DNN with weight layer W and α_S the parameters of the same DNN but with weight layer S. This means that the random matrix R does not improve the accuracy of the DNN as $N \to \infty$.

More generally, we also show that

$$\mathbb{P}\Big(\delta X(s, \alpha_S) \geq 0 \mid \delta X(s, \alpha_W) \geq a(N)\|s\|_2\Big) \geq 1 - \frac{1}{N^{1/4}} \tag{C.1}$$

with $a(N) = 2/N^{1.5/4}$.

We start by approximating the components of Rs and showing that they are small and go to 0 as $N \to \infty$. When R is a random matrix with components taken from i.i.d. with mean 0 variance $1/N$, we have that $(Rs)_m$ is a random variable taken from a distribution with mean 0 and variance

$$\sigma^2_{(Rs)_m} = \sum_{i=1}^{M} s_i^2 \frac{1}{N} = \|s\|_2^2 \frac{1}{N}.$$

Thus, $(Rs)_m$ is also a random variable with 0 mean and variance $\|s\|_2^2/N$.

Then, using Chebyshev's inequality (see Theorem C.1) and taking $k = N^{0.5/4}$ we obtain

$$\Pr\left(|(Rs)_m| \geq \frac{1}{N^{1.5/4}}\|s\|_2\right) \leq \frac{1}{N^{1/4}}. \tag{C.2}$$

Thus, given that

$$\begin{aligned}\delta X(s, \alpha_W) - \delta X(s, \alpha_S) &= \left|(R+S)s_{i(s)}\right| - \left|\max_{j \neq i(s)} \left((S+R)s\right)_j\right| - \left|(S)s_{i(s)}\right| + \left|\max_{j \neq i(s)} (Ss)_j\right| \\ &\leq \left|(R)s_{i(s)}\right| - \left|\max_{j \neq i(s)} \left((S+R)s\right)_j\right| + \left|\max_{j \neq i(s)} (Ss)_j\right|. \end{aligned} \tag{C.3}$$

By (C.2), we have

$$\begin{aligned}&\mathbb{P}\left(\left(\delta X(s, \alpha_W) - \delta X(s, \alpha_S)\right) \leq \frac{1}{N^{1.5/4}}\|s\|_2 - \left|\max_{j \neq i(s)} ((S+R)s)_j\right| + \left|\max_{j \neq i(s)} (Ss)_j\right|\right) \\ &\geq 1 - \frac{1}{N^{1/4}}. \end{aligned} \tag{C.4}$$

Suppose $|\max_{j\neq i(s)}(Ss)_j|$ is satisfied for the component $k^*(N)$, meaning that

$$\left|\max_{j\neq i(s)}(Ss)_j\right| = \left|(Ss)_{k^*(N)}\right|. \tag{C.5}$$

Then, again by (C.2) we have

$$\mathbb{P}\left(\left|\left|\left((S+R)s\right)_{k^*}\right| - \left|(Ss)_{k^*}\right|\right| \leq \frac{1}{N^{1.5/4}}\|s\|_2\right) \geq 1 - \frac{1}{N^{1/4}}, \tag{C.6}$$

given that S and R are independent from each other and S is deterministic. Given that

$$\max_{j\neq i(s)}(R+S)s_j \geq (R+S)s_{k^*},$$

from (C.4) we obtain

$$\mathbb{P}\left(\left(\delta X(s,\alpha_W) - \delta X(s,\alpha_S)\right) \leq \frac{1}{N^{1.5/4}}\|s\|_2 - \left|\left((S+R)s\right)_{k^*}\right| + \left|(Ss)_{k^*}\right|\right) \geq 1 - \frac{1}{N^{1/4}}. \tag{C.7}$$

Final result from (C.6):

$$\mathbb{P}\left(\delta X(s,\alpha_W) - \delta X(s,\alpha_S) \leq \frac{2}{N^{1.5/4}}\|s\|_2\right) \geq 1 - \frac{1}{N^{1/4}} \tag{C.8}$$

with $a(N) = 2/N^{1.5/4}$.

When the DNN has more than one layer, we continue this proof as follows:

$$\begin{aligned} Z = \mathbb{P}\big(&\|\lambda\circ W_4\circ\lambda\circ W_3\circ\lambda\circ(R+S)\circ\lambda\circ W_1 s \\ &- \lambda\circ W_4\circ\lambda\circ W_3\circ\lambda\circ S\circ\lambda\circ W_1 s\|_1 > t\big). \end{aligned} \tag{C.9}$$

By the triangle inequality, we have

$$Z \leq \mathbb{P}\big(\|W_4\circ\lambda\circ W_3\circ\lambda\circ(R+S)\circ\lambda\circ W_1 s - W_4\circ\lambda\circ W_3\circ\lambda\circ S\circ\lambda\circ W_1 s\|_1 > t\big). \tag{C.10}$$

We factor out the common term W_4 and use the inequality $\|Av\|_1 \leq \|A\|_1\|v\|_1$ to obtain

$$Z \leq \mathbb{P}\big(\|W_4\|_1\cdot\|W_3\circ\lambda\circ(R+S)\circ\lambda\circ W_1 s - W_3\circ\lambda\circ S\circ\lambda\circ W_1 s\|_1 > t\big). \tag{C.11}$$

Next, we apply the inequality $\|Av\|_1 \leq \|A\|_1\|v\|_1$ on W_3 and $\lambda\circ(R+S)\circ\lambda\circ W_1 s - \lambda\circ S\circ\lambda\circ W_1 s$ together with the triangle inequality, we get

$$Z \leq \mathbb{P}\big(\|W_4\|_1\cdot(\|W_3\|_1\cdot\|RW_1 s\|_1) > t\big), \tag{C.12}$$

given that

$$\|\lambda\circ(R+S) - \lambda\circ S\| \leq \|R\|.$$

Thus we have,

$$Z \leq \mathbb{P}\left(\|W_4\|_1\cdot\left(\|W_3\|_1\cdot\max_{i,j}|R_{i,j}|\,\|W_1 s\|_1\right) > t\right). \tag{C.13}$$

If R is a random matrix with i.i.d. entries taken from the normal distribution $N(0, 1/N)$, then using the concentration inequality in Theorem C.2, we get: If we choose

$$t = \|W_4\|_1\|W_3\|_1\|W_1 s\|_1 \left(\frac{1}{N^{1.5/4}} + \sqrt{\frac{2\log N^2}{N}}\right),$$

then

$$Z \leq \mathbb{P}\left(\max_{i,j} |R_{i,j}| > \frac{1}{N^{1.5/4}} + \sqrt{\frac{2\log N^2}{N}}\right) \leq 2\exp\left(-\frac{N^{1/4}}{2}\right).$$

Thus, we have

$$\mathbb{P}\left(\delta X(s, \alpha_{S_2}) \geq 0 \mid \delta X(s, \alpha_{W_2}) > \|W_4\|_1\|W_3\|_1\|W_1 s\|_1 \left(\frac{1}{N^{1.5/4}} + \sqrt{\frac{2\log N^2}{N}}\right)\right)$$
$$\leq 2\exp\left(-\frac{N^{1/4}}{2}\right). \tag{C.14}$$

For a DNN with more layers and a different R, the steps in this proof would be the same and would use similar concentration inequalities to bound $\max_{i,j} |R_{i,j}|$. □

Proof of the Pruning Theorem 4.1. We first provide a proof for the case when our DNN has only one layer matrix W.

By Lemma 4.2 we have that there exists $D(N)$ such that for

$$E := a(N)h_\phi(s, b) + b(N), \tag{C.15}$$

we have the conditional probability

$$\mathbb{P}\big(\delta X(s, \alpha_S) \geq 0 \mid \delta X(s, \alpha_W) \geq E\big) \geq 1 - D(N) \tag{C.16}$$

with $D(N), a(N), b(N) \to 0$ as $N \to \infty$ and $h_\phi(s, b)$ coming from (4.11). Again, α_S are the parameters of the DNN with the weight matrix S and α_W are the parameters of the DNN with the weight layer matrix W.

We then use Approximation Lemma B.2 to obtain that there exists f_W, and exists $B_N^*(\epsilon)$ such that for any ϵ,

$$\mathbb{P}\big(\|(W' - S)z\|_2 > (1 + \epsilon)f_W\|z\|_2\big) < B_N^*(\epsilon) \tag{C.17}$$

with $B_N^*(\epsilon) \to 0$ as $N \to \infty$.

We then follow an argument similar to that given for Lemma E.1. That is, the change in classification confidence due to pruning is

$$\Delta(\delta X) = |\delta X(s, \alpha_S) - \delta X(s, \alpha_{W'})|.$$

For a particular component i, the change ΔX_i due to pruning is given by

$$\Delta X_i = |X_i(s, \alpha_S) - X_i(s, \alpha_{W'})|.$$

We have that $X(s, \alpha_S) = \lambda \circ (W)s$ and $X(s, \alpha_{W'}) = \lambda \circ (W')s$.

Given that λ is the absolute value activation function, for any scalar values x and y, we have

$$|\lambda(x) - \lambda(y)| \leq |x - y|. \tag{C.18}$$

Thus, from (C.17) we have

$$\|X(s, \alpha_S) - X(s, \alpha_{W'})\| \leq \|(S - W')s\| \leq (1 + \epsilon) f_W \|s\|_2$$

with probability $1 - B_N^*(\epsilon)$. Furthermore,

$$\Delta X_{\max} + \Delta X_{\max -1} \leq \sqrt{2}(1 + \epsilon) f_W \|s\|_2 \tag{C.19}$$

with probability $1 - B_N^*(\epsilon)$. Here $\Delta X_{\max -1}$ is the change in the component of X which has the second to biggest change, and $\Delta X_{\max}$ is the change in the change in the component of X which had the biggest change.

Then, using the same steps given in Lemma E.1, we obtain

$$\Delta(\delta X) \leq \Delta X_{\max} + \Delta X_{\max -1} \leq \sqrt{2}(1 + \epsilon) f_W \|s\|_2 \tag{C.20}$$

with probability $1 - B_N^*(\epsilon)$. Thus, given that

$$\mathbb{P}\big(\delta X(s, \alpha_S) \geq 0 \mid \delta X(s, \alpha_W) \geq E\big) \geq 1 - D(N), \tag{C.21}$$

we have that there exists an explicit function $f_W > 0$ such that for any $\epsilon > 0$ there exists $C_\epsilon(N)$ such that for

$$E' := (1 + \epsilon)\big(\sqrt{2}(1 + \epsilon) \min\{f_W, \sqrt{\lambda_+}\} g_\phi(s, b) + a(N) h_\phi(s, b) + b(N)\big), \tag{C.22}$$

we have the conditional probability

$$\mathbb{P}\big(\delta X(s, \alpha_{W'}) \geq 0 \mid \delta X(s, \alpha_W) \geq E'\big) \geq 1 - C_\epsilon(N) \tag{C.23}$$

with $C_\epsilon(N), a(N), b(N) \to 0$ as $N \to \infty$ and $g_\phi(s, b)$ coming from (4.11). f_W is given in Lemma B.2. When the DNN has more than one layer, we continue this proof with the same steps found in Section E.1. □

Theorem C.1 (Chebyshev's Inequality). *Let X be a random variable with mean μ and finite non-zero variance σ^2. Then for any real number $k > 0$,*

$$\Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}. \tag{C.24}$$

Borell-TIS inequality for i.i.d. Gaussian variables

Theorem C.2 (Borell-TIS Inequality). *Let $X_1, X_2, \ldots, X_n$ be i.i.d. centered Gaussian random variables with $X_i \sim N(0, \sigma^2)$. Set*

$$s_X^2 := \max_{i=1,\ldots,n} \mathbb{E}\big(X_i^2\big) = \sigma^2.$$

Then for each $t > 0$

$$P\left(\max_{i=1,\ldots,n}|X_i| - \mathbb{E}\Big[\max_{i=1,\ldots,n} X_i\Big] > t\right) \leq \exp\left(-\frac{t^2}{2s_X^2}\right).$$

For the absolute value bound

$$P\left(\Big|\max_{i=1,\ldots,n}|X_i| - \mathbb{E}\Big[\max_{i=1,\ldots,n} X_i\Big]\Big| > t\right) \leq 2\exp\left(-\frac{t^2}{2s_X^2}\right).$$

In particular, if $X_i \sim N(0, 1/N)$,

$$s_X^2 = \frac{1}{N},$$

and for any $t > 0$,

$$P\left(\max_{i=1,\ldots,n}|X_i| > \sqrt{\frac{2\log n}{N}} + t\right) \leq 2\exp\left(-\frac{Nt^2}{2}\right).$$

C.2 Proof of Pruning Theorem for accuracy

This theorem follows from the Pruning Theorem 4.1. Under the assumptions of this theorem and from the Pruning Theorem, and assuming that R_2 is i.i.d. from $N(0, 1/N)$ we have that for any $\epsilon > 0$, for any $s \in T'$,

$$\begin{aligned}\mathbb{P}\Big(\delta X(s, \alpha_{W'}) \geq 0 \mid \delta X(s, \alpha_W) \geq (1+\epsilon)\big(\sqrt{2}\big(f_{W_2}\sigma_{\max}(W_3) + 2N^{-\frac{1.5}{4}}\|W_3\|_1\big)\big) \\ \times \max_{s \in G_{E_{acc},\alpha}} \|W_1 s\|_1 + b(N)\Big) \\ \geq 1 - C_\epsilon(N) \qquad \text{(C.25)}\end{aligned}$$

with $C_\epsilon(N), b(N) \to 0$ as $N \to \infty$ and f_{W_2} given by the Approximation Lemma B.2. In this case, E' from the Pruning Theorem given in (4.18) is equal to $(1+\epsilon)(\sqrt{2}(f_{W_2}\sigma_{\max}(W_3) + 2N^{-1.5/4}\|W_3\|_1))\max_{s \in G_{E_{acc},\alpha}}\|W_1 s\|_1 + b(N)$. By taking

$$E_{acc} = (1+\epsilon)\Big(\sqrt{2}\big(f_{W_2}\sigma_{\max}(W_3) + 2N^{-\frac{1.5}{4}}\|W_3\|_1\big)\Big)\max_{s \in G_{E_{acc},\alpha}}\|W_1 s\|_1 + b(N), \qquad \text{(C.26)}$$

we make the classification confidence threshold of the non-pruned DNN independent on s. We then obtain, from (C.25), that for all s,

$$\mathbb{P}\big(\delta X(s, \alpha_{W'}) \geq 0 \mid \delta X(s, \alpha_W) \geq E_{acc}\big) \geq 1 - C_\epsilon(N). \qquad \text{(C.27)}$$

Given that this is independent on s, we obtain the final result

$$\mathbb{P}\big(G_{E_{acc},\alpha} \subset G_{0,\alpha'}\big) \geq \big(1 - C_\epsilon(N)\big)^{|G_{E_{acc},\alpha}|}. \qquad \text{(C.28)}$$

The theorem then follows from the fact that $acc_{\alpha'}(t) = |G_{0,\alpha'}|/|T'|$.

Appendix D Other algorithms required for implementing RMT-SVD based pruning of DNN

D.1 BEMA algorithm for finding λ_+

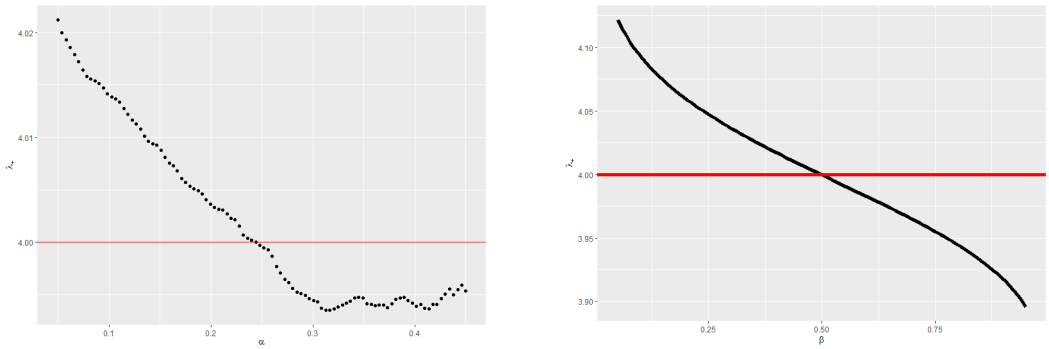
The following is the BEMA algorithm for finding the best fit λ_+ of $R^\top R/N$ based on the ESD of $X = R + S$. It is used in the analysis of matrices with the information plus noise structure (i.e. deformed matrices), where one wants to determine the rightmost edge of the compact support of the MP distribution. In this context, the Tracy-Widom distribution provides the limiting distribution of the largest eigenvalue λ_+ of large random matrices, allowing us to compute a confidence interval for λ_+ in the presence of the Marčenko-Pastur distribution, see [28]. The BEMA algorithm is computationally efficient and has been shown to provide accurate results for matrices with the information plus noise structure. More details on the algorithm and its relationship with the MP and Tracy-Widom distributions can be found in [28]. Here, we present a simplified version of the algorithm for R a $N \times N$ matrix.

Algorithm 2 Computation of λ_+ Using MP and Tracy-Widom Distributions.

1: Choose parameters $\alpha \in (0, 1/2), \beta \in (0, 1)$.
2: **for** each $\alpha N \leq k \leq (1-\alpha)N$ **do**
3: Obtain q_k, the (k/N) upper-quantile of the MP distribution with $\sigma^2 = 1$ and $c = 1$. ▷ Each q_k is a solution to $\int_0^{q_k} \frac{1}{2\pi} \frac{\sqrt{(4-\lambda)\lambda}}{\lambda} = k/N$.
4: **end for**
5: Compute $\hat{\sigma}^2 = \frac{\sum_{\alpha N \leq k \leq (1-\alpha)N} q_k \lambda_k}{\sum_{\alpha N \leq k \leq (1-\alpha)N} q_k^2}$. ▷ where λ_k is the k-th smallest eigenvalue of X.
6: Obtain $t_{1-\beta}$, the $(1-\beta)$ quantile of Tracy-Widom distribution.
7: Return $\lambda_+ = \hat{\sigma}^2[4 + 2^{4/3} t_{1-\beta} \cdot N^{-2/3}]$.

Remark D.1. The algorithm depends on parameters $\alpha \in (0, 1/2), \beta \in (0, 1)$. We show this by varying α and β for the case found in Example D.1. See Fig. D.1(a) and D.1(b). The red line is $\lambda_+ = 4$, which is the correct λ_+ of $R^\top R/N$. In this example, while dependence on α is insignificant for sufficiently large values, dependence on β allows us to control the confidence that the eigenvalues of the random matrix R will be smaller than the estimator for λ_+ of the MP distribution. In all the numerical simulations given in Section 3, we took $\alpha = 0.1$ and $\beta = 0.1$. It would be interesting to try and see what happens when we take a larger β, as it would prevent the algorithm from pruning too many parameters but might lead to even higher accuracy.

In the numeric portion of the paper, we always divide $R^\top R$ by $1/N$ when obtaining the ESD of X regardless of the original distribution of the initial random matrix $R(0)$. If



(a) Dependence of algorithm the choice of α, $\beta = 0.5$. In this example, the rank of the deterministic matrix S is fairly low.

(b) Dependence of algorithm on the choice of β, $\alpha = 0.25$.

Figure D.1: How λ_+ depends on α and β.

$R(o)$ is distributed using $N(0, 1/N)$, dividing $R^\top R$ by N does not seem to change the fact that the ESD of X is given by the MP distribution.

Example D.1. In this example, we create a random $N \times N$ matrix R with components taken from i.i.d. using the normal distribution of zero mean and unit variance ($\sigma^2 = 1$). We take S to be a $N \times N$ deterministic matrix with components given by

$$S[i,j] = \tan\left(\frac{\pi}{2} + \frac{1}{j+1}\right) + \cos(i) \cdot \log(i+j+1) + \sin(j) \cdot \cos\left(\frac{i}{j}\right), \tag{D.1}$$

$W = R + S$ and $X = W^\top W/N$. The BEMA algorithm is used to find the λ_+ of the ESD of X, as described in Section D.1. R is a random matrix satisfying the conditions of Theorem 3.1, and so the ESD of $1/NR^\top R$ converges to the Marchenko-Pastur distribution as $N \to \infty$ and has a λ_+ that determines the rightmost edge of its compact support. We can imagine a situation in which R is not directly known, and the goal is to find an estimator of λ_+ from the ESD of X. See Fig. D.2 for the result of the ESD of X with the Marchenko-Pastur distribution that best fits the ESD shown in red.

The bulk of the eigenvalues are well-fit by the MP distribution, but some eigenvalues bleed out to the right of λ_+. These eigenvalues correspond to the singular values of S. The direct calculation of the λ_+ of the MP distribution corresponding to $R^\top R/N$ gives

$$\lambda_+ = \sigma^2 \cdot (1+1)^2 = 4,$$

and the λ_+ obtained to fit the bulk of the ESD of X and the λ_+ of $R^\top R/N$ are approximately the same.

The BEMA algorithm will be employed to estimate λ_+ from the ESD of $X_l(t)$. As the DNN training progresses, it is expected that the majority of the eigenvalues of $X_l(t)$ will conform to the MP distribution. Nonetheless, some eigenvalues may extend beyond the bulk of the MP distribution and be associated with the singular values of $S_l(t)$. The purpose of the BEMA algorithm is to identify the furthest edge of the MP distribution,

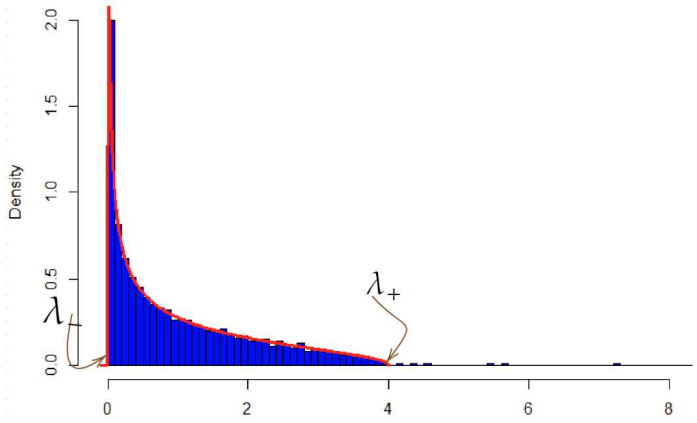


Figure D.2: In blue, we have the ESD of X; in red, the Marchenko-Pastur distribution, which best fits the ESD based on the BEMA algorithm.

which helps determine the value of λ_+. Understanding λ_+ is crucial as it offers insights into the DNN's behavior during training and its capacity to generalize to new data.

In combination with the SVD, the BEMA algorithm can be applied to decide which singular values of the DNN's weight matrices W_l should be eliminated during training. The SVD breaks down the weight matrix into its singular values and singular vectors, allowing for an RMT-based analysis of their distribution. Utilizing the BEMA algorithm, one can pinpoint the eigenvalues associated with the singular values of S_l and differentiate them from the eigenvalues related to the singular values of R_l. By removing the eigenvalues corresponding to R_l, the DNN's training process can be made more effective and efficient.

D.2 The role of singular value decomposition in deep learning

Consider a $N \times M$ matrix A. A singular value decomposition of A consists of a factorization $A = U\Sigma V^\top$, where

- U is an $N \times N$ orthogonal matrix.
- V is an $M \times M$ orthogonal matrix.
- Σ is an $N \times M$ matrix with the ith diagonal entry equal to the ith singular value σ_i and all other entries of Σ being zero.

For λ_i, the eigenvalues of a matrix $X = W^\top W$, the singular values of W are given by $\sigma_i = \sqrt{\lambda_i}$. Consequently, singular values are connected to eigenvalues of the symmetrization of a matrix W.

For a DNN's W_l, it has been demonstrated that discarding small singular values of W_l through its SVD during the DNN's training can decrease the number of parameters while improving accuracy, as shown in [3, 11, 62, 63]. In the remainder of this work, we illustrate how RMT can aid in identifying the singular values to be removed from a DNN layer without compromising the DNN's accuracy.

In particular, the BEMA algorithm can be combined with the SVD of W_l to ascertain which singular values should be removed during the DNN's training. To achieve this, one first computes the SVD of W_l and then calculates the eigenvalues of the symmetrized matrix $X_l = W_l^\top W_l / N$. The eigenvalues derived from the symmetrization can be linked to the singular values of W_l through $N\lambda_i = \sigma_i^2$. Employing the BEMA algorithm to estimate the value of λ_+ allows for the determination of a threshold for the singular values of W_l. Singular values below the threshold can be removed without impacting the DNN's accuracy, as they are likely less crucial for the DNN's performance. This process can be carried out iteratively during the DNN's training since the threshold can be updated as training advances.

D.3 Eliminating singular values while preserving accuracy

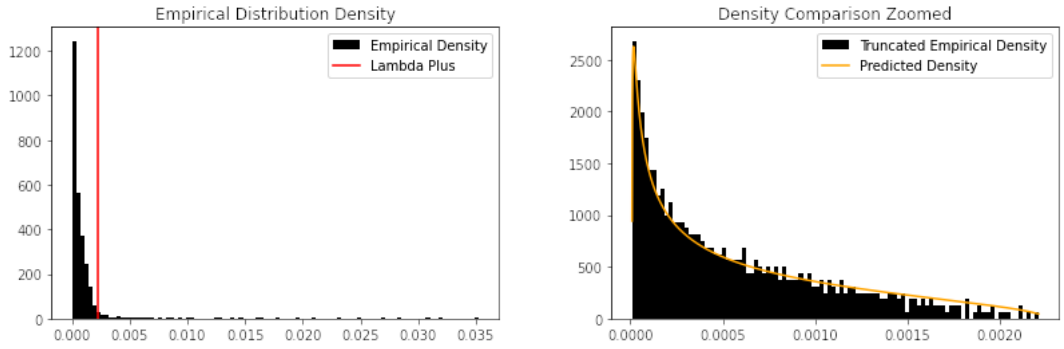
In this subsection, we demonstrate how SVD can be employed to remove the random components of W_l without compromising accuracy. This could potentially lead to a significant reduction in the number of parameters in the DNN, resulting in faster training.

Algorithm 3 Pruning a Weight Matrix from a Trained DNN.

1: Acquire a weight matrix W_l from a trained DNN.
2: Perform SVD on W_l: $W_l = U\Sigma V^\top$.
3: Calculate the eigenvalues λ_i of the square matrix $W_l^\top W_l / N$.
4: Apply the BEMA algorithm from Section D.1 to find the best fit MP distribution for the ESD of $X = W_l^\top W_l / N$ and its corresponding λ_+. ▷ See Fig. D.3.
5: Determine whether the ESD of X fits the MP distribution using the algorithm in Section D.4. ▷ Ensures $W_l = R + S$ assumption is valid.
6: Replace a portion, e.g. 0.1, of the singular values less than $\sqrt{\lambda_+ N}$ with zeros to form a new diagonal matrix Σ' and the truncated matrix W_l'.
7: Use Σ' to obtain $W_{1,l}' = U\sqrt{\Sigma'}$ and $W_{2,l}' = \sqrt{\Sigma'}V^\top$.

Example D.2. Consider an original DNN with two hidden layers, each consisting of 10 nodes. The total number of parameters in this case would be 100. By employing SVD and removing 8 small singular values in the weight layer matrix of this DNN, we can split the hidden layer into two, resulting in a new DNN with three hidden layers. The first layer will have 10 nodes, the second layer will have 2 nodes, and the third layer will have 10 nodes. By keeping only two singular values in the SVD, we now have only 20 parameters, see Fig. D.4. In practice, we do not actually split the layer.

Example D.3. We used the above approach for a DNN trained on MNIST. In this example, the DNN has two layers, the first with a 784×1000 matrix W_1 and the second with a 1000×10 matrix W_2. The activation function was ReLU. We trained the DNN for 10 epocs and achieved a 98% accuracy on the test set.



(a) Full Empirical Density (b) Zoomed Density

Figure D.3: The ESD of X_l and its best fit MP distribution.

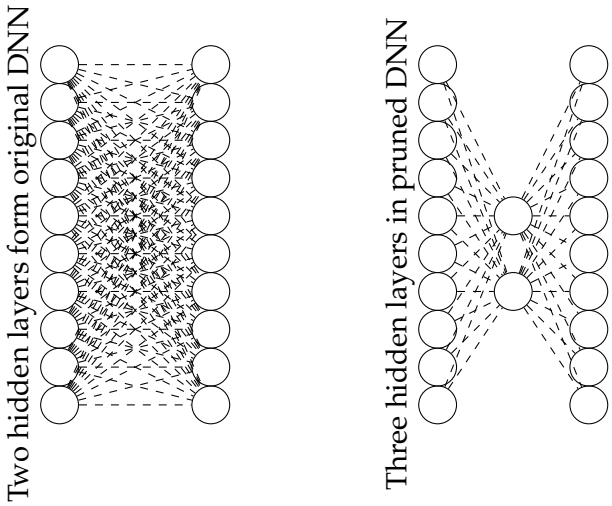


Figure D.4: Two hidden layers from the original, in the left figure, have 10 nodes (total 100 parameters). Layers are transformed into three hidden layers, in the right figure, in pruned DNN. The first layer has 10 nodes, the second layer has 2 nodes (keeping only two singular values in the SVD), and the third layer has 10 nodes, resulting in a total of 20 parameters.

We perform an SVD on W_1, in this case Σ is a 784×1000 matrix. Even if we only keep the biggest 60 σ_i of W_1 and transform the first layer into two layers $W_{1,1}$ and $W_{2,1}$ the accuracy is still 98%. W_1 had $784,000$ parameters, while $W_{1,1}$ and $W_{2,1}$ have $784(60) + 1,000(60) = 107,040$ parameters (not including the bias vector parameters). This is a reduction of over 85%. In Fig. D.5 we show how the accuracy of the DNN depends on the number of singular values that we keep. The red line corresponds to the threshold given by the MP distribution (via λ_+) for how many of the large singular values should be kept. As the figure shows, this threshold is highly accurate. This example also numerically confirms Theorem 4.1 and shows that the threshold given in the theorems (for which singular values to keep) is highly useful and accurate.

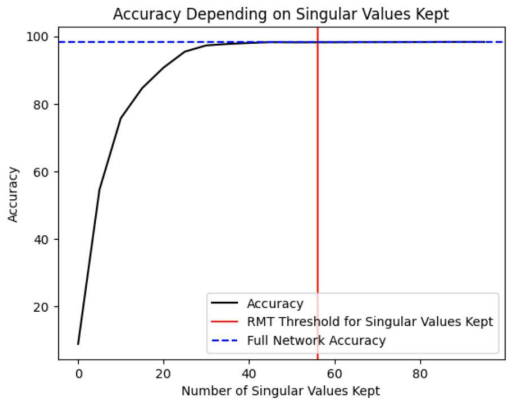


Figure D.5: Number of eigenvalues kept is shown on the x-axis while the accuracy is shown on the y-axis.

D.4 MP fit criteria: Checking if the ESD of X fits a MP distribution

This subsection details a procedure to evaluate whether the ESD of a square matrix X is possibly drawn from a specific MP distribution (with a possibility of spiked eigenvalues). The initial step of this procedure relies on the BEMA method to identify the most fitting MP distribution. This optimal fitting distribution provides a theoretical cumulative distribution function (CDF), while the empirical cumulative spectral distribution related to X can be computed. A comparison of these two distributions allows us to dismiss the hypothesis that X follows the suggested MP distribution if the difference between the distributions is substantial. Let us formalize these concepts, starting with the concept of an empirical cumulative spectral distribution.

Definition D.1. *Assume G is an $N \times M$ matrix and its ESD μ_{G_M} is defined as in Definition 3.1. The empirical cumulative spectral distribution of G, symbolized as $F_G : \mathbb{R} \to \mathbb{R}$, is defined as*

$$F_G(a) = \mu_{G_m}\big((-\infty, a]\big). \tag{D.2}$$

Interestingly, the cumulative distribution functions for the MP distribution are known and can be expressed in a closed form. With these equations, we can comprehensively describe our procedure. We set a tuning parameter $\gamma \in (0, 1)$ corresponding to the sensitivity of our test.

This procedure computes the maximum difference between the expected and empirical cumulative distribution functions by sampling at each point in the empirical distribution. Since this is intended for the unique case of testing for spiked MP distributions, we can utilize this information to enhance our test over simply calculating the L^∞ difference between the expected and empirical distributions.

This enhancement is reflected in the step which calculates i_{low} and i_{high}. As BEMA only uses data in the quantile between $(\alpha, 1-\alpha)$ to find the best fit, it is logical to only examine for fit within the same range. In this context, we would expect a spiked MP distribution to be poorly approximated by its generative MP distribution around the highest eigenvalues (i.e. the spiked values), and hence it makes sense to only test the bulk values for goodness of fit.

Algorithm 4 Assessing Conformance to the MP Distribution.

1: Accept $X = W^\top W/N$ as input, where W is an $N \times M$ matrix.
2: Calculate the spectrum of $X = \{\sigma_1, \ldots, \sigma_M\}$.
3: Compute the empirical cumulative spectral distribution of X, denoted F_X.
4: Execute the BEMA method with parameters α and β to determine $\hat{\sigma}^2$, the anticipated variance of each coordinate of W.
5: Calculate $0 \leq i_{\text{low}} < i_{\text{high}} \leq M$ such that i_{low} is the smallest integer with $i_{\text{low}}/M \geq \alpha$ and i_{high} is the largest integer with $i_{\text{high}}/M \leq 1 - \alpha$.
6: Define F'_X as the theoretical cumulative distribution function for the MP distribution with parameters $\hat{\sigma}^2$ and $\lambda = N/M$.
7: Evaluate $s = \max_{i \in [i_{\text{low}}, i_{\text{high}}]} |F_X(i) - F'_X(i)|$.
8: **if** $s > \gamma$ **then**
9: Dismiss the hypothesis that X follows the proposed distribution.
10: **else**
11: Do not reject this hypothesis.
12: **end if**

Appendix E Some of the proofs and numerics

E.1 Proof of Lemma 4.1

Proof. The classification confidence before pruning is

$$\delta X(s, \alpha_{W_b}) = X_{i(s)}(s, \alpha_{W_b}) - \max_{j \neq i(s)} X_j(s, \alpha_{W_b}).$$

After pruning, it is

$$\delta X\big(s, \alpha_{W'_b}\big) = X_{i(s)}\big(s, \alpha_{W'_b}\big) - \max_{j \neq i(s)} X_j\big(s, \alpha_{W'_b}\big).$$

For simplicity, we will start by proving the theorem for the case of a DNN with only one layer matrix W and a bias vector β. Thus, we take $X(s, \alpha_{W_b}) = X(s, \alpha_W)$ and $X(s, \alpha_{W'_b}) = X(s, \alpha_{W'})$.

Then, the change in classification confidence due to pruning is

$$\Delta(\delta X) = |\delta X(s, \alpha_W) - \delta X(s, \alpha_{W'})|.$$

For a particular component i, the change ΔX_i due to pruning is given by

$$\Delta X_i = |X_i(s, \alpha_W) - X_i(s, \alpha_{W'})|.$$

We have that $X(s, \alpha_W) = \lambda \circ (W + \beta)s$ and $X(s, \alpha_{W'}) = \lambda \circ (W' + \beta)s$.

Given that λ is the absolute value activation function, for any scalar values x and y, we have

$$|\lambda(x) - \lambda(y)| \leq |x - y|. \tag{E.1}$$

Thus

$$\|X(s,\alpha_W) - X(s,\alpha_{W'})\| \leq \|(W-W')s\| \leq \sqrt{\lambda_+}\|s\|_2.$$

If the change in the norm of the entire output vector due to pruning is at most $\sqrt{\lambda_+}\|s\|_2$, then the maximum change in any individual component must also be bounded by that amount. That is

$$\Delta X_{\max} \leq \sqrt{\lambda_+}\|s\|_2.$$

Furthermore,

$$\Delta X_{\max} + \Delta X_{\text{max-1}} \leq \sqrt{2\lambda_+}\|s\|_2 \tag{E.2}$$

with $\Delta X_{\text{max-1}}$ the change in the component of X which has the second to biggest change and $\Delta X_{\max}$ the change component of X which had the biggest change.

Now assume

$$\max_{j\neq i(s)} X_j(s,\alpha_{W'}) \leq \max_{j\neq i(s)} X_j(s,\alpha_W),$$

then given that

$$|X_{i(s)}(s,\alpha_W) - X_{i(s)}(s,\alpha_{W'})| \leq \sqrt{\lambda_+}\|s\|_2,$$

we must have

$$\delta X(s,\alpha_{W'}) = X_{i(s)}(s,\alpha_{W'}) - \max_{j\neq i(s)} X_j(s,\alpha_{W'}) \leq \sqrt{\lambda_+}\|s\|_2.$$

Next, assume

$$\max_{j\neq i(s)} X_j(s,\alpha_{W'}) \geq \max_{j\neq i(s)} X_j(s,\alpha_W).$$

Let us expand the change in δX due to pruning

$$\begin{aligned}\Delta(\delta X) &= \left|\left(X_{i(s)}(s,\alpha_W) - \max_{j\neq i(s)} X_j(s,\alpha_W)\right) - \left(X_{i(s)}(s,\alpha_{W'}) - \max_{j\neq i(s)} X_j(s,\alpha_{W'})\right)\right| \\ &\leq |X_{i(s)}(s,\alpha_W) - X_{i(s)}(s,\alpha_{W'})| + \left|\max_{j\neq i(s)} X_j(s,\alpha_W) - \max_{j\neq i(s)} X_j(s,\alpha_{W'})\right|.\end{aligned}$$

Take k^* to be an integer such that

$$\max_{j\neq i(s)} X_j(s,\alpha_{W'}) = X_{k^*}(s,\alpha_{W'}),$$

we have, given that

$$\max_{j\neq i(s)} X_j(s,\alpha_{W'}) \geq \max_{j\neq i(s)} X_j(s,\alpha_W),$$

that

$$\begin{aligned}&|X_{i(s)}(s,\alpha_W) - X_{i(s)}(s,\alpha_{W'})| + \left|\max_{j\neq i(s)} X_j(s,\alpha_W) - \max_{j\neq i(s)} X_j(s,\alpha_{W'})\right| \\ &\leq |X_{i(s)}(s,\alpha_W) - X_{i(s)}(s,\alpha_{W'})| + |X_{k^*}(s,\alpha_{W'}) - X_{k^*}(s,\alpha_W)|.\end{aligned} \tag{E.3}$$

Consider the worst-case scenario where the change is maximally concentrated in the components $i(s)$ and k^*. Thus, given (E.2) we have

$$\Delta(\delta X) \leq \Delta X_{\max} + \Delta X_{\text{max-1}} \leq \sqrt{2\lambda_+}\|s\|_2. \tag{E.4}$$

Next, we consider the case where the DNN has multiple layers given by the matrices $W_1, \cdots, W_L$, and bias vectors $\beta_1, \cdots, \beta_L$ and we prune the last matrix layer of the DNN, i.e. we prune layer matrix W_L to obtain W_L'. In this case, we can reduce this problem to that of a DNN with a single layer matrix W_L and a single bias vector β_L and an input vector

$$z = \lambda \circ W_{L-1} \circ \cdots \circ \lambda \circ W_1 s.$$

Then, by the above argument, we have

$$\Delta(\delta X) \leq \sqrt{2\lambda_+}\|\lambda \circ (W_{L-1} + \beta_{L-1}) \circ \cdots \circ \lambda \circ (W_1 + \beta_1)s\|_2. \tag{E.5}$$

Finally, we assume that we prune a layer W_b to obtain W_b'. By a similar argument to what we have above, we see that the max change of a component of the output vector $X(s, \alpha_{W_b})$ after pruning is

$$\Delta X_{\max} \leq \sqrt{\lambda_+}\|\lambda \circ (W_{b-1} + \beta_{b-1}) \circ \cdots \circ \lambda \circ (W_1 + \beta_1)s\|_2 \sigma_{\max}(W_{b+1}) \ldots \sigma_{\max}(W_L),$$

given that

$$\|\lambda \circ (Az + \beta) - \lambda \circ (Az' + \beta)\|_2 \leq \sigma_{\max}(A)\|z' - z\|_2.$$

Again, we obtain,

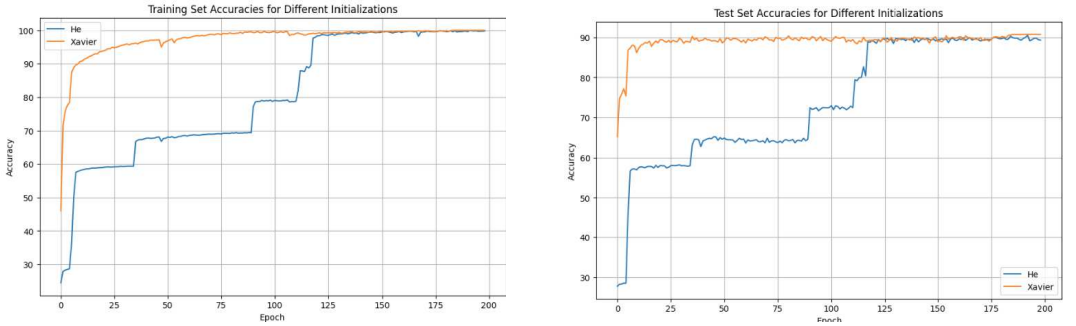
$$\begin{aligned}\Delta X_{\max} + \Delta X_{\text{max-1}} \leq \sqrt{2\lambda_+}&\|\lambda \circ (W_{b-1} + \beta_{b-1}) \circ \cdots \circ \lambda \circ (W_1 + \beta_1)s\|_2 \\ &\times \sigma_{\max}(W_{b+1}) \ldots \sigma_{\max}(W_L).\end{aligned} \tag{E.6}$$

Thus, this completes the proof of the lemma. □

E.2 Effectiveness of MP-based pruning for different initialization methods

In this subsection, we investigated the performance of various initialization methods, including He [24] and Xavier [21] initialization. It is important to note that both the He and Xavier initializations align with the principles of the MP theorem given in Theorem 3.1. That is, the weight layer components are i.i.d. with mean zero and bounded variance. In practice, for the weight layer matrices W initialized based on those distributions, we have that the ESD of $W^\top W$ fits the MP distribution well (with an error, see Section D.4, of ~ 0.001 for both). Thus, the Pruning Theorem 4.1 would hold for both of these initializations.

The DNNs were fully connected, and their architecture was given by $[784, 3000, 3000, 3000, 3000, 500, 10]$. When pruning with MP-based pruning, both initializations achieved test accuracies above 90%, which was comparable to the performance of networks initialized with a normal distribution (which was 90.74%), see Fig. E.1. Without MP-based pruning, the DNNs achieved accuracy on the test set of $\sim$ 89%. All the DNNs were trained



(a) Training Set Accuracy vs Epoch for He and Xavier Initializations (b) Test Set Accuracy vs Epoch for He and Xavier Initializations

Figure E.1: Accuracy vs Epoch for He and Xavier Initializations.

using a combination of $L2$ and $L1$ regularization (see (E.7)), which is why the DNNs achieved an accuracy above 90%. The using of both $L1$ and $L2$ regularization together with MP-pruning is a critical factor in achieving these high accuracies, consistently above 90%. The interplay between MP-based pruning and regularization, which contributes to this performance, will be discussed in detail in another paper. We used the hyperparameters $\mu_1 = 0.0000005$ and $\mu_2 = 0.0000001$. The other hyperparameters are the same as those given in the numerical simulations from Section 3.2.1.

$$L\big(\alpha(a)\big) = -\frac{1}{|T|}\sum_{s\in T}\log\Big(\phi_{i(s)}\big(s,\alpha(a)\big)\Big) + \mu_1\sum_{i=1}^{L}\|W_i(a)\|_1 + \mu_2\sum_{i=1}^{L}\|W_i(a)\|_F^2. \tag{E.7}$$

Additionally, we conducted a similar simulation where the initial weights were drawn from a normal distribution $N(0, 1/N)$, but 90% of the parameters were randomly initially sparsified afterward and set to zero (though we still used them during training). The resulting DNN's test accuracy plateaued at $\sim$ 80% and failed to improve beyond this point, even when using MP-based pruning. It is important to note that the MP theorem does not hold for weight layers initialized in such a manner. Similar results were found when we used He and Xavier initializations but initially sparsified them so that 90% of the parameters started out as zero (but were used during training).

Finally, we also ran the numerical simulations for the above DNN architecture and hyperparameters, but setting $\mu_1 = 0$ (that is only using $L2$ regularization, which is what was also done in Section 3.2.1). For the He and Xavier initialization, the DNNs, with MP-based pruning, obtained an accuracy of $\sim$ 90%, comparable to the simulations found in Section 3.2.1, while for the sparse initialization, their accuracy never appreciated higher than 80% even when using MP-based pruning. Without MP-based pruning, the He and Xavier initialization accuracies were $\sim$ 88%.

E.3 A regression problem: MP-based pruning in regression

We consider the task of finding a DNN that approximates a function best fitting the given data in terms of Mean Squared Error (MSE), see Fig. E.2.

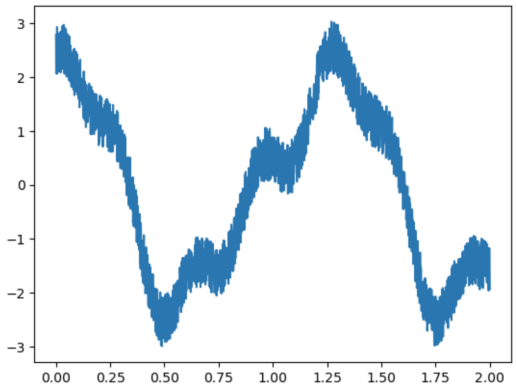


Figure E.2: Illustration of the regression problem.

Training data. The training data consists of 2000 equally spaced points between 0 and 2

$$x_{\text{train}} = \left\{ x_i \,\middle|\, x_i = \frac{2i}{1999},\ i = 0, 1, 2, \ldots, 1999 \right\}.$$

Noise is added to the training labels. The noise is uniformly distributed between -1 and 1 with a scaling factor of $\sigma = 0.5$,

$$\text{noise}_{\text{train}} = \{\epsilon_i \mid \epsilon_i \sim \mathcal{U}(-1, 1)\}.$$

The training labels are generated using the following function:

$$y_{\text{train}} = 0.5\cos(20x_{\text{train}}) + 2\cos(5x_{\text{train}}) + 0.5\sin(10x_{\text{train}}) + \sigma \cdot \text{noise}_{\text{train}}.$$

Testing data. The testing data consists of 500 equally spaced points between 0 and 2

$$x_{\text{test}} = \left\{ x_i \,\middle|\, x_i = \frac{2i}{499},\ i = 0, 1, 2, \ldots, 499 \right\}.$$

The testing labels are generated using the following function without noise:

$$y_{\text{test}} = 0.5\cos(20x_{\text{test}}) + 2\cos(5x_{\text{test}}) + 0.5\sin(10x_{\text{test}}).$$

Defining output and target. The DNN used for this 2D regression has a simple structure:

- The first and last layers each consist of a single neuron.
- The network outputs a single value and is trained using MSE together with $L2$ regularization as the loss function.

DNN topology. We use a fully connected DNN for the regression problem given in Fig. E.2, that is, to find the curve that best fits the data. The fully connected DNN used in this regression problem has the following topology: $[1, 1000, 1000, 1000, 1000, 1]$, which represents the number of nodes in each layer. We do not use an activation function after the final layer of the DNN.

Training process. Both an unpruned DNN and a DNN using RMT pruning were trained to minimize MSE loss. The hyperparameters for training can be found in Table E.1. We decreased the lr by 0.997 every epoch.

Table E.1: Training hyperparameters for the DNN.

Hyperparameter	Value
Number of epochs	1198
Number of seeds	5
Learning rate (lr)	0.025
Momentum	0.95
Batch size	128
$L2$ regularization on loss	0.001

Comparison of unpruned vs pruned DNN. The pruned DNN shows a significantly smaller loss on both training and test sets, with a 50% reduction in the number of parameters, see Figs. E.3 and E.4. It is important to note that a smaller DNN might perform better on this task when MP-based pruning is not employed. However, for larger fully connected DNNs it is clear that MP-based pruning helps the DNN achieve much lower loss for this task.

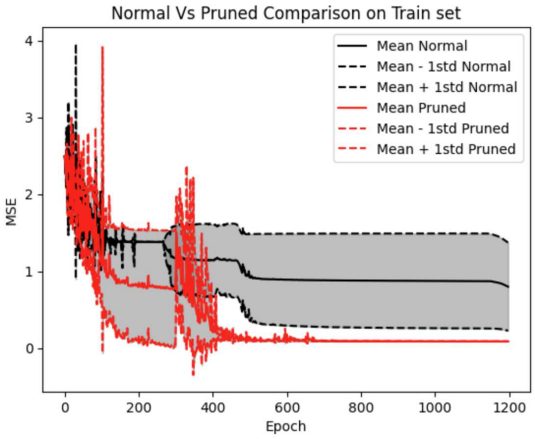


Figure E.3: Training loss comparison between normal and pruned DNN.

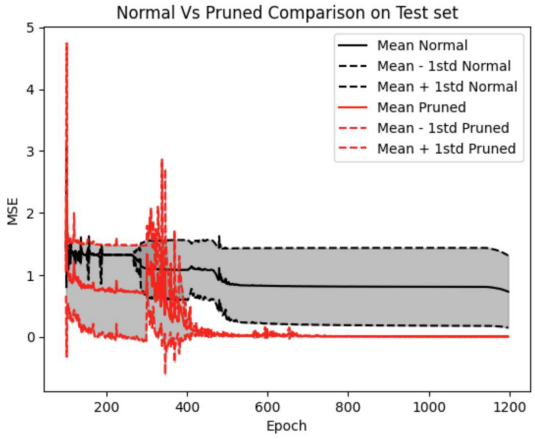


Figure E.4: Test loss comparison between normal and pruned DNN.

E.4 Numerical example used to calculate δX.

In this demonstration, we have a set of two-dimensional data points, and our objective is to identify the boundary that divides them into two categories. The dataset originates from a randomly constructed polynomial function of a designated degree. We then sample data points uniformly across a spectrum of x-values. For every x-value, the poly-

nomial function gives a y-value. These points are then slightly offset either upwards or downwards, forming two distinguishable point clusters labeled as red and blue. Points positioned above the polynomial curve get a blue label, and those below are tagged red. We also add Gaussian noise to slightly modify the y-values, causing a few red data points to appear below the boundary and some blue ones above. This scenario is depicted in Fig. E.5.

The goal is to harness a DNN to capture the decision boundary demarcating the two clusters. This DNN is designed to process a two-dimensional data point, producing a binary output indicating whether the point is red or blue. Given that the dataset is artificially curated, the actual decision boundary is known, allowing us to gauge the efficacy of our DNN.

For this task, our neural network model had one hidden layer with 500 neurons. We also used the ReLU activation function. Training is executed using the cross-entropy loss complemented by the SGD optimizer and momentum.

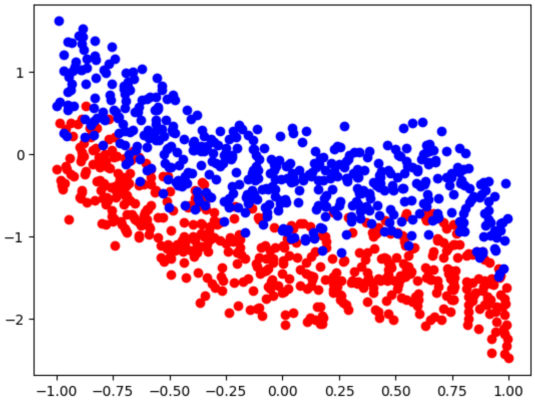


Figure E.5: Illustration of a decision boundary for a binary classification challenge created using a random polynomial function, supplemented with noise. The distinct blue and red points signify the two classes.

E.5 Hyperparameters for Section 3.2.1

- Number of epochs: 40.
- Number of seeds: 10.
- Learning rate (lr): 0.02.
- Momentum: 0.9.
- Batch size: 128.
- $L2$ regularization on loss: 0.0005. (The regularization term is applied for both DNNs, the normal and pruned versions).

See [26, 33, 55] for information on learning rate, momentum, batch size, and regularization, respectively.

E.6 Hyperparameters for Section 3.2.1

- Number of epochs: 70.
- Number of seeds: 10.
- Learning rate (lr): 0.02.
- Momentum: 0.9.
- Batch size: 128.
- $L2$ regularization on loss: 0.0005.

E.7 Hyperparamters for Section 3.2.2

Training hyperparameters:

- Number of epochs: Depends on the example.
- Number of seeds: 5.
- Learning rate (lr): 0.02.
- Momentum: 0.9.
- Batch size: 128.
- Split frequency: Depends on the example.
- L^2 regularization on loss: 0.001.
- Goodness of fit: Changes for every simulation.

E.8 CNN architecture description

The CNN model in this study consists of multiple convolutional and fully connected layers, incorporating batch normalization and dropout for regularization. The architecture is designed as follows:

Convolutional layers:

- The network contains several convolutional layers with kernels of size 3. Each convolutional layer is followed by batch normalization.
- Activation functions (ReLU) are applied conditionally based on predefined configurations.
- Max-pooling layers are introduced after every second convolutional layer, except the first one. Specifically, max-pooling with a pool size of 2 is used to reduce the spatial dimensions of the feature maps.
- Dropout is applied after each convolutional layer to prevent overfitting, with a dropout rate of 0.35.

Fully connected layers:

- After the convolutional layers, the output is flattened and passed through a series of fully connected layers.
- Each fully connected layer is followed by batch normalization and, conditionally, by a ReLU activation function based on the configuration.
- Dropout is also applied to the output of the first fully connected layer to further prevent overfitting.

Output layer:

- The final layer applies a log-softmax function to produce the output probabilities for classification.

E.8.1 Pooling and regularization details

Pooling layers: Max-pooling layers are strategically placed to downsample the feature maps, specifically after every second convolutional layer. This pooling strategy helps in reducing the computational complexity and in extracting invariant features.

Batch normalization: Batch normalization is applied after each convolutional and fully connected layer to stabilize and accelerate the training process by normalizing the input to each layer.

Dropout: Dropout is utilized after each convolutional layer and the first fully connected layer with rates of 0.35 and variable values, respectively, to reduce overfitting by randomly setting a fraction of activations to zero during training.

E.9 The hyperparamters for Section 3.2.3

Training hyperparameters:

- Number of epochs: 300.
- Number of seeds: 5.
- Learning rate (lr): 0.001.
- Momentum: 0.9.
- Batch size: 128.
- Split frequency (every how many epochs we split the pruned DNN and remove small singular values): 40.
- $L2$ regularization on loss: 0.001.
- goodness of fit: 0.08 for fully connected layers, 0.06 for convolutional layers.

Acknowledgments

The authors thank J. Tanner for bringing several relevant publications to our attention.

The work of L. Berlyand was partially supported by the NSF (Grant Nos. DMS-2005262 and IMPRESS-U 2401227). L. Berlyand and E. Sandier are grateful to the Labex Bézout Foundation for supporting the stay of LB while visiting Université Paris-Est, which helped facilitate the collaboration between E. Sandier, L. Berlyand, and Y. Shmalo on this work.

References

[1] H. Abdi and L. J. Williams, Principal component analysis, *Wiley Interdiscip. Rev. Comput. Stat.*, 2(4):433–459, 2010.

[2] J. Agterberg, Z. Lubberts, and C. E. Priebe, Entrywise estimation of singular vectors of low-rank matrices with heteroskedasticity and dependence, *IEEE Trans. Inf. Theory*, 68(7):4618–4650, 2022.

[3] X. Anhao, Z. Pengyuan, P. Jielin, and Y. Yonghong, SVD-based DNN pruning and retraining, *Tsinghua. Sci. Technol.*, 56(7):772–776, 2016.

[4] J. Baik, G. Ben Arous, and S. Péché, Phase transition of the largest eigenvalue for nonnull complex sample covariance matrices, *Ann. Probab.*, 33(5):1643–1697, 2005.

[5] Z. Bao, X. Ding, and K. Wang, Singular vector and singular subspace distribution for the matrix denoising model, *Ann. Statist.*, 49(1):370–392, 2021.

[6] Z. Bao and D. Wang, Eigenvector distribution in the critical regime of BBP transition, *Probab. Theory Related Fields*, 182(1-2):399–479, 2022.

[7] F. Benaych-Georges and R. R. Nadakuditi, The eigenvalues and eigenvectors of finite, low rank perturbations of large random matrices, *Adv. Math.*, 227(1):494–521, 2011.

[8] F. Benaych-Georges and R. R. Nadakuditi, The singular values and vectors of low rank perturbations of large rectangular random matrices, *J. Multivariate Anal.*, 111:120–135, 2012.

[9] L. Berlyand, P.-E. Jabin, and C. A. Safsten, Stability for the training of deep neural networks and other classifiers, *Math. Models Methods Appl. Sci.*, 31(11):2345–2390, 2021.

[10] R. Bro and A. K. Smilde, Principal component analysis, *Anal. methods*, 6(9):2812–2831, 2014.

[11] C. Cai, D. Ke, Y. Xu, and K. Su, Fast learning of deep neural networks via singular value decomposition, in: *Pacific Rim International Conference on Artificial Intelligence*, Springer, 820–826, 2014.

[12] Y. Chen, C. Cheng, and J. Fan, Asymmetry helps: Eigenvalue and eigenvector analyses of asymmetrically perturbed low-rank matrices, *Ann. Statist.*, 49(1):435, 2021.

[13] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, The loss surfaces of multilayer networks, in: *Artificial Intelligence and Statistics*, PMLR, 192–204, 2015.

[14] R. Couillet and M. Debbah, *Random Matrix Methods for Wireless Communications*, Cambridge University Press, 2011.

[15] R. Couillet and Z. Liao, *Random Matrix Methods for Machine Learning*, Cambridge University Press, 2022.

[16] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, *Adv. Neural Inf. Process. Syst.*, 4:2933–2941, 2014.

[17] P. Dharmawansa, P. Dissanayake, and Y. Chen, The eigenvectors of single-spiked complex wishart matrices: Finite and asymptotic analyses, *IEEE Trans. Inf. Theory*, 68(12):8092–8120, 2022.

[18] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, Efficient and accurate estimation of Lipschitz constants for deep neural networks, *Adv. Neural Inf. Process. Syst.*, Vol. 32, 2019.

[19] J. Ge, Y.-C. Liang, Z. Bai, and G. Pan, Large-dimensional random matrix theory and its applications in deep learning and wireless communications, *Random Matrices: Theory and Applications*, 10(04):2230001, 2021.

[20] R. Ge, F. Huang, C. Jin, and Y. Yuan, Escaping from saddle points – online stochastic gradient for tensor decomposition, in: *Conference on Learning Theory*, PMLR, 797–842, 2015.

[21] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 249–256, 2010.
[22] G. H. Golub and C. F. van Loan, Matrix computations, *SIAM Review*, 28(2):252–255, 1986.
[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.
[24] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: *2015 IEEE International Conference on Computer Vision*, IEEE, 1026–1034, 2015.
[25] G. Hinton et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal. Process Mag.*, 29(6):82–97, 2012.
[26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, *arXiv:1207.0580*, 2012.
[27] S. Hochreiter and J. Schmidhuber, Flat minima, *Neural Comput.*, 9(1):1–42, 1997.
[28] Z. T. Ke, Y. Ma, and X. Lin, Estimation of the number of spiked eigenvalues in a covariance matrix by bulk eigenvalue matching analysis, *J. Am. Stat. Assoc.*, 118:374–392, 2021.
[29] H. Kösters and A. Tikhomirov, Limiting spectral distributions of sums of products of non-Hermitian random matrices, *arXiv:1506.04436*, 2015.
[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Commun. ACM*, 60(6):84–90, 2017.
[31] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, Handwritten digit recognition with a back-propagation network, *Adv. Neural Inf. Process. Syst.*, Vol. 2, 1989.
[32] W. E. Leeb, Matrix denoising for weighted loss functions and heterogeneous signals, *SIAM J. Math. Data Sci.*, 3(3):987–1012, 2021.
[33] A. Lewkowycz, Y. Bahri, E. Dyer, J. Sohl-Dickstein, and G. Gur-Ari, The large learning rate phase of deep learning: The catapult mechanism, *arXiv:2003.02218*, 2020.
[34] M. W. Mahoney and C. H. Martin, Traditional and heavy tailed self regularization in neural network models, in: *International Conference on Machine Learning*, PMLR, 4284–4293, 2019.
[35] V. A. Marchenko and L. A. Pastur, Distribution of eigenvalues for some sets of random matrices, *Sb. Math.*, 114(4):507–536, 1967.
[36] C. H. Martin and M. W. Mahoney, Heavy-tailed universality predicts trends in test accuracies for very large pre-trained deep neural networks, in: *Proceedings of the 2020 SIAM International Conference on Data Mining*, SIAM, 505–513, 2020.
[37] C. H. Martin and M. W. Mahoney, Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning, *J. Mach. Learn.*, 22(1):7479–7551, 2021.
[38] C. H. Martin, T. Peng, and M. W. Mahoney, Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data, *Nat. Commun.*, 12(1):4122, 2021.
[39] X. Meng and J. Yao, Impact of classification difficulty on the weight matrices spectra in deep learning and application to early-stopping, *J. Mach. Learn.*, 24:1–40, 2023.
[40] A. Naumov, V. Spokoiny, and V. Ulyanov, Bootstrap confidence sets for spectral projectors of sample covariance, *Probab. Theory Related Fields*, 174(3):1091–1132, 2019.
[41] S. O'rourke, V. Van, and K. Wang, Matrices with Gaussian noise: Optimal estimates for singular subspace perturbation, *arXiv:1803.00679*, 2018.
[42] S. O'Rourke, V. Vu, and K. Wang, Random perturbation of low rank matrices: Improving classical bounds, *Linear Algebra Appl.*, 540:26–59, 2018.
[43] J. Park, I. Pelakh, and S. Wojtowytsch, Minimum norm interpolation by perceptra: Explicit regularization and implicit bias, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2023.
[44] R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, in: *International Conference on Machine Learning*, PMLR, 1310–1318, 2013.
[45] L. Pastur, On random matrices arising in deep neural networks. Gaussian case, *arXiv:2001.06188*, 2020.
[46] L. Pastur and V. Slavin, On random matrices arising in deep neural networks: General I.I.D. case, *Random Matrices: Theory and Applications*, 12(01):2250046, 2023.
[47] L. Prechelt, Early stopping – but when? in: *Neural Networks: Tricks of the Trade. Lecture Notes in Computer*

Science, 53–67, 2012.
[48] I. Price and J. Tanner, Dense for the price of sparse: Improved performance of sparsely initialized networks via a subspace offset, In *International Conference on Machine Learning*, PMLR, 8620–8629, 2021.
[49] M. Ringnér, What is principal component analysis? *Nat. Biotechnol.*, 26(3):303–304, 2008.
[50] T. N. Saada and J. Tanner, On the initialisation of wide low-rank feedforward neural networks, *arXiv:2301.13710*, 2023.
[51] V. I. Serdobolskii, *Multivariate Statistical Analysis: A High-Dimensional Approach*, Springer Science & Business Media, 2000.
[52] Y. Shmalo, J. Jenkins, and O. Krupchytskyi, Deep learning weight pruning with RMT-SVD: Increasing accuracy and reducing overfitting, *arXiv:2303.08986*, 2023.
[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.
[54] M. Staats, M. Thamm, and B. Rosenow, Boundary between noise and information applied to filtering neural network weight matrices, *arXiv:2206.03927*, 2022.
[55] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, On the importance of initialization and momentum in deep learning, in: *International Conference on Machine Learning*, PMLR, 1139–1147, 2013.
[56] I. Sutskever, O. Vinyals, and Q. V. Le, Sequence to sequence learning with neural networks, *Adv. Neural Inf. Process.*, Vol. 27, 2014.
[57] M. Thamm, M. Staats, and B. Rosenow, Random matrix analysis of deep neural network weight matrices, *Phys. Rev. E*, 106(5):054124, 2022.
[58] S. Vadera and S. Ameen, Methods for pruning deep neural networks, *IEEE Access*, 10:63280–63300, 2022.
[59] R. Vershynin, *High-Dimensional Probability*, Cambridge University Press, 2018.
[60] X. Xiao, Z. Li, C. Xie, and F. Zhou, Heavy-tailed regularization of weight matrices in deep neural networks, *arXiv:2304.02911*, 2023.
[61] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, W. Dai, Y. Qi, Y. Chen, W. Lin, and H. Xiong, Trained rank pruning for efficient deep neural networks, in: *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, IEEE, 14–17, 2019.
[62] J. Xue, J. Li, and Y. Gong, Restructuring of deep neural network acoustic models with singular value decomposition, *Proc. Interspeech*, 2365–2369, 2013.
[63] H. Yang, M. Tang, W. Wen, F. Yan, D. Hu, A. Li, H. Li, and Y. Chen, Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, IEEE, 678–679, 2020.
[64] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, Understanding deep learning (still) requires rethinking generalization, *Commun. ACM*, 64(3):107–115, 2021.
[65] Z. Zhang and G. Pan, Tracy-Widom law for the extreme eigenvalues of large signal-plus-noise matrices, *arXiv:2009.12031*, 2020.