Data Science Semester Project

Signal Processing Laboratory (LTS4)

Sources of Redundancy in Neural Networks

Author:
Julien Niklas Heitmann

Supervisor:
Guillermo Ortiz Jiménez

Head of Laboratory:
Pascal Frossard

Content

1 Introduction

One of the suprising results of training deep neural networks with many more parameters than training samples is that one can achieve zero-training loss but still get good generalisation (Neyshabur et al. 2018). While statistical learning theory suggests that such heavily over-parameterized networks generalize poorly without further regularization, experiments show that in most cases, increasing the number of parameters does not worsen the capability to generalize. In fact, it often happens that, when training two models with $N1$ and $N2$ neurons respectively, $N1 > N2$, training will converge to zero-training loss in both cases, but the larger model will generalize better. The question arises why the network doesn't exploit its full expressive power to overfit the training dataset. Modern network architectures, which can have up to 100x more trainable parameters than training samples (Zagoruyko and Komodakis 2016), most certainly are capable of learning the entire dataset independently of the labels. It has been shown that zero training loss can even be achieved when the training labels are randomly shuffled (Zhang et al. 2016), thus preventing the learning of underlying features of the data that are sufficient to solve the classification task at hand.

Surely one must look at the interplay between the dataset properties, the network architecture and the optimization algorithm, but the latter in particular seems to play a crucial role, as it somehow consistently favors solutions in the optimization landscape that generalize, over solutions that overfit the data and in some cases do not learn a representation of the data. Is it a property of the optimization algorithm, does stochastic gradient descent optimization lead to sparse solutions? If yes, how do these sparse solutions look like? Do trained neural networks rely on single components (nodes or filters), even with a growing number of parameters? Or does each of the components contribute equally to the estimated function, thus reducing the contribution when there are more parameters?

An interesting way to measure the importance of a trained network's individual components is pruning. When single components are removed from a network's architecture, i. e. they do not contribute anymore to the output of the network, the difference in validation accuracy can be measured, which is a good indicator of the network's performance. Pruning can therefore be used as a means to get a better understanding of what is happening in a trained network, to identify important parts and not so important ones. But pruning can also be seen as an end, if done properly it might yield smaller architectures, which result in computation speed-ups and smaller memory footprints at evaluation. Assuming that over-parameterization is necessary to achieve good performance and generalization, and that some components of an over-parameterized network are obsolete after training or can be removed without a negative impact on validation accuracy if the network is retrained, then pruning might even be necessary to get efficient architectures that scale.

This project aims to explore different hypotheses about neural network training dynamics, with a particular focus on sources of redundancy at the level of neurons and convolutional filters. The goal is to identify which sources of redundancy could be exploited to design new structured pruning strategies. A visualization tool will be introduced to help get a better intuition for the complex process that is neural network training. Moreover, an analysis of the space spanned by the rows of chosen layer weight matrices will be done, to confirm observations and get more rigorous results.

Background on network pruning

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, and a desired sparsity level κ (i. e., the number of non-zero weights), unstructured pruning can be written as the following constrained optimization problem:

$$\min_\theta L(\theta; \mathcal{D}) = \min_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(\theta; (\mathbf{x}_i, \mathbf{y}_i)),$$
$$\text{s.t. } \theta \in \mathbb{R}^m, \|\theta\|_0 \leq \kappa.$$

Here, $\ell(\cdot)$ is the standard loss function (e. g. cross-entropy loss), θ is the set of parameters of the neural network and m is the total number of parameters.

When taking a weight-vector perspective, we are dealing with units such as neurons or convolutional filters, that have an associated weight vector. Our model is a feed-forward network with $N + 1$ layers, corresponding to N hidden layers and one output layer. At a given layer $L, 1 \leq L \leq N + 1$, the weights of the layer can be grouped into a matrix $\mathbf{W}^L$, where the dimensions of the weight matrix depends on the type of the layer (convolutional or fully connected). To be able to theoretically manipulate all layer weight matrices in a similar way, we reshape the convolutional weight matrices by flattening all but the last dimension. Conceptually, all layer weight matrices are $m^L \times n^L$ matrices for us, where m^L, n^L are the number of units at layer L and the length of the weight vectors at layer L, respectively. The notation $\mathbf{W}$ will be used throughout this report to refer to a generic layer weight matrix.

Structured pruning removes entire weight vectors of a layer, instead of individual weights. Let us apply a mask to all layer weight matrices, that is replace $\mathbf{W}$ by $\mathbf{MW}$, where $\mathbf{M} \in \mathbb{R}^{m \times m}$ is a diagonal matrix with diagonal $\mathbf{m} \in \mathbb{R}^m$, whose entries lie in $\{0, 1\}$. Then structured pruning, when applied to a certain layer, minimizes the loss function with the constraint that $\|\mathbf{m}\|_0 \leq \kappa$. Thus κ is the number of non-zero units. Structured pruning can be applied to multiple layers at a time.

2 Related work

If good generalization can be achieved by an overparameterized network, there must be some mechanism that prevents the model to overfit the dataset, and learn for instance features that would be considered as noise when it comes to the classification task, because they do not provide any information about a sample's affiliation to a certain class. With an increasing number of parameters, more complex decision boundaries can be represented, so in order to generalize, the solutions obtained by the optimization algorithm of choice must be "simple" in a way, which is yet to be defined. While it is curious that large neural networks do not overfit the dataset, even in the presence of massive label noise (Rolnick et al. 2017), it is even more surprising that increasing the number of parameters can decrease the generalization gap (Neyshabur et al. 2018). In other words, given a classification task and a certain model architecture with a fixed number of layers but adjustable layer widths, a certain degree of layer-wise over-parameterization might be both sufficient and necessary to achieve good performance after training.

2.1 Self-regularization of over-parameterized models

The inability of statistical learning to explain and predict the properties of neural networks is not a new phenomenon. In particular, when it comes to measuring complexity, traditional measures such as the VC dimension or the number of parameters fail to explain why increasing the number of parameters of a model does not necessarily increase the generalization gap. More recently new measures have been introduced that aim to take into account the self-regularizing effect observed during the training of large models. For instance, Neyshabur et al. 2018 measure a network's capacity by looking at the Frobenius norm of the difference of the hidden layer weights with the initialization, and empirically show that this measure decreases with increasing network size. This suggests that the bigger the network, the smaller the work the optimization algorithms needs to do, since the hidden layer weights are closer to their initialization.

Furthermore, one should keep in mind when examining a model's capacity, that the notion of overparameterization itself is very vague, as it is intractable to measure a classification problem's complexity, and therefore determine the appropriate number of parameters. Experiments involving teacher-student models avoid this problem and are thus easier to analyze. The teacher is a model with a specific architecture and fixed weights, that given a random input produces an output. This generative process is repeated multiple times, and the input-output pairs are given to a student model as training data. The student is a model with a similar architecture, but a larger number of hidden units, and the goal for the student is to learn the teacher's model. Because the student can express much more complex functions than the teacher function it has to learn, it is said to be over-parameterized. Goldt et al. 2019 found evidence that, with a teacher-student setup, upon over-parameterization Stochastic Gradient Descent finds solutions which amount to performing an effective model averaging, thus improving robustness and avoiding overfitting.

This property seems to extend to more practical problems than the artificial teacher-student setup, there has been empirical evidence that supports that repeated units emerge when increasing a network's width, as a way to adjust the capacity of the model (Casper et al. 2019). Those repeated units have highly correlated outputs, implying that similar activation regions were learned. Additionally, removable units appear, which can be dropped out of the network without significantly hurting the validation accuracy. This kind of self-regularization enforces capacity constraints, as the function modeled by a network with either of these units could me modeled by a simpler one.

The emergence of repeated or removable units during the training process of large models indicates that the self-regularizing mechanism, that allow overparameterized networks to generalize after training, opperate at least partly at the level of individual units. There has been work that makes use of recent advances in Random Matrix Theory (RMT) to describe the different phases of that implicit self-regularization of the weight matrices (Martin and Mahoney 2018). More specifically, the Empirical Spectral Density (ESD) of the correlation matrix $\mathbf{X} = \mathbf{W}\mathbf{W}^T$ associated with the layer weight matrix $\mathbf{W}$ is analysed over the course of training. RMT is used to identify different phases of training using models to describe the shape of each ESD. In each phase the layer weight matrix $\mathbf{W}$ is modeled as "noise plus signal", where the noise is modeled by a random matrix $\mathbf{W}^{rand}$, which entries are drawn from a normal distribution, and the signal is a correction Δ^{sig}, $\mathbf{W} \simeq \mathbf{W}^{rand} + \Delta^{sig}$. The empirically

verified models suggest that during training, $\mathbf{W}^{rand}$ decreases, and Δ^{sig}, a low-rank matrix, increases in magnitude. The hypothesis that, when training an overparameterized network, only a low-rank subspace of the weight matrices is relevant for the learning target has also been put forward by Nagarajan and Kolter 2019. To support their claims they analyze the singular value distribution of the update matrix $\mathbf{W} - \mathbf{Z}$, which is the difference between the final weight matrix $\mathbf{W}$ and the weight matrix at initialization $\mathbf{Z}$. They show that the validation accuracy is able to endure the heavy pruning process of projecting the update matrix onto its top 16 left-singular vectors (out of 1024), and conclude that all but a few top singular directions can be considered as noise. They also notice that the spectral norm of the matrix that is removed when projecting onto the top singular vectors (the residuals) can have quite a high spectral norm. These results highlight the fact that low-complexity solutions are found when training networks that have a high representational power and, that even in the presence of noise introduced by SGD, the low-complexity component of the solution dominates the output of the network.

2.2 Advantageous properties of over-parameterized models

After looking at capacity constraining mechanisms in overparameterized neural networks, there is yet another, more intriguing question that needs to be answered: why can overparameterized neural networks achieve better generalization than smaller networks, even in cases such as a teacher-student setup where the smaller network is a student whose architecture matches that of the teacher (Tian et al. 2019)? In an attempt to provide an answer to that question, a closer look will be taken at two theories that motivated some of the first steps of the project. The first one, more formaly known as the "Lottery ticket hypothesis" (Frankle and Carbin 2018), emphasises the importance of weight initialization, which leads to the formation of subnetworks that can be trained efficiently. The second one states that overparameterization works well because of better feature exploration and weight clustering (Brutzkus and Globerson 2019).

The Lottery Ticket Hypothesis

Contemporary experience suggests that overparameterized networks are easier to train, and achieve better generalization. But such networks can be pruned, which sometimes heavily reduces the parameter-count (Han et al. 2015). One might ask why we do not train instead architectures discovered by pruning. It is commonly believed that these pruned networks are more difficult to train from scratch, and reach lower accuracy than the fine-tuned pruned networks (Li et al. 2016). The "Lottery ticket hypothesis" (Frankle and Carbin 2018) is a theory that provides an explanation as to why over-parameterized networks might perform better, and has gained a lot of popularity recently. It states the following:

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

According to the authors, at initialization, some weights that form a sub-network "win the lottery", because the combination of their initial value and the way they are arranged in that particular sub-network makes them particularly "trainable" by the chosen optimization

algorithm, compared to other weights that are not in the sub-network. Those weights form what is called a "winning ticket", and the paper provides a way to identify the winning ticket after multiple iterations of pruning, based on the weight magnitudes, and retraining, to compensate for the loss in accuracy caused by pruning. One of the important results of the paper is that when the weights of the winning ticket, which can be seen as a mask applied to the weights of the network, are reset to their initial value (all other weights set to zero), then the newly obtained network reaches similar if not better accuracy than the original network, up to a certain degree of pruning (up to 96%). Surprisingly, this result does not hold when the weights of the winning ticket are re-sampled, meaning that both the pruned architecture and the values of the unpruned weights are of importance. When randomly re-sampled, the winning tickets learn slower and reach lower accuracy than the re-initialized one.

Note that, as individual weights are pruned to identify the winning ticket, the theory makes use of unstructured pruning, and a winning ticket obtained by this procedure would have to rely on specialized libraries and hardware to exploit benefits of the weight matrix sparsity. According to Liu et al. 2018, the results do not hold when using structured pruning, and might even be misleading in the case of unstructured pruning. The authors of the paper claim that the optimization algorithm (ADAM) and the small initial learning rate of the original paper lead to inferior accuracy when the winning ticket is randomly initialized, but this can be remedied using SGD and a higher learning rate. Moreover, as claimed in the original paper, the procedure to identify a winning ticket fails for deeper networks. Nonetheless, according to Frankle and Carbin, even if in some cases, up to a certain level of sparsity, highly overparameterized networks can be pruned, reinitialized and retrained successfully, beyond a certain point, when the network is extremely pruned, less severely overparameterized networks only maintain a good validation accuracy if they are well-initialized. Also, winning tickets have been found for deeper networks (Frankle, Dziugaite, et al. 2019) when, instead of resetting the weights of the winning ticket to their initial values, they are rewinded to their former values at iteration k, where k is much smaller than the total number of training iterations. Finally, winning tickets generate so much interest because they might reveal a lot about how to better initialize neural networks, but also help better understand the bias of modern optimization algorithms towards sparse solutions. Surprisingly, it has been shown that, within the natural images domain, winning ticket initializations generalize across a variety of datasets, often achieving performance close to that of winning tickets generated on the same dataset (Morcos et al. 2019). This suggests that winning tickets somehow have an inductive bias generic to neural networks, which generalizes to multiple configurations.

Alignment of weight vectors during training

Brutzkus and Globerson 2019 define a simple 3-layer neural network for a binary classification task they call the XORD problem (XOR Detection). Its input is a vector with $2d$ entries, which take values in $\{-1, 1\}$, and the vector can be written as a sequence of d pairs. The goal is to determine whether any of the pairs has twice the same value (1 or -1). For this purpose a neural network is trained, with a convolutional, a max-pool and a fully connected layer. Only the weights of the convolutional layer, which are vectors in $\mathbb{R}^2$, are trainable. The authors prove that increasing the size of the convolutional layer results in better feature exploration, and that after convergence the weight vectors associated to the learned convolutions cluster around just a few directions. Furthermore, they show that

when the size of the convolutional layer is small (just a few trainable weights), the network can achieve zero-training error on specific datasets, but not generalize well, because it is missing some patterns. This is less likely to happen with a larger convolutional layer because of better exploration of the feature space.

According to the authors, this might explain why over-parameterized neural networks don't overfit the training dataset and in some cases even generalize better than smaller networks. The former can be explained by weight clustering, which reduces the effective capacity of the network, and the latter by better exploration of the input space. Hence the observed generalization properties of over-parameterized networks do not contradict the findings of statistical learning, since training with gradient descent will facilitate the formation of clusters, thus reducing the network's expressive power. Empirically, the paper looks at a 3-layer neural network with a large convolutional layer, trained on MNIST. More specifically, after training, the vectors associated to the convolutional filters are clustered using K-Means with four clusters. The distribution of angles with respect to the closest cluster center is compared to that of clustered untrained vectors (at initialization). The findings are in accordance with the XORD theory, as trained vectors tend to be much closer to the center of the cluster they are part of.

2.3 Leveraging model simplicity: Pruning and Compression

We have seen that it is often desirable to train large networks, because of the implicit mechanisms that prevent them to fully exploit their expressive power and the advantageous properties. The problem with ever-growing models is that they come with increased requirements of computational resources (memory size, number of computation operations, power usage), which is problematic in resource-constrained settings. While we might not be able to get rid of the expensive process that is training a large model, because of the better generalization properties after convergence, we might be able to reduce its size once trained, since we know it doesn't make use of its full capacity and could therefore be compressed. Many methods have been developed to address this issue, and just a few of them will be discussed in this report.

The goal of network pruning is, given a large reference model, to learn a smaller subnetwork that mimics the performance of the reference model. At the most fine-grained level of network pruning, there is individual weight pruning, that removes connections between units of a network based on certain criteria. This is often referred to as *unstructured pruning*, because it can lead to very sparse weight matrices with no direct efficiency gains when not used in combination with specialized hardware or libraries. The idea to remove individual weights to compress a network dates back to Optimal Brain Damage (LeCun, Denker, and Solla 1990) and Optimal Brain Surgeon (Hassibi and Stork 1993), that make use of the Hessian of the loss function to determine a connection's saliency, which is the decrease in test accuracy when the weight is eliminated. More recent methods rely on magnitude to evaluate a weight's saliency, pruning weights that have a small magnitude. This computationally inexpensive method has shown to work for larger methods and became the de facto standard method for network pruning (e. g. used in the Lottery Ticket Hypothesis to identify winning tickets). As mentioned, the benefits of unstructured pruning cannot be directly exploited, since the original components of the network (convolutional filters and

neurons) are still preserved.

Structured pruning doesn't have this problem as it prunes at the level of neurons, filters or even layers, which directly reduces the size of the network without the need for specialized hardware or libraries to evaluate the network's output. Magnitude-based methods are also used when performing structured pruning, Li et al. 2016 prune convolutional channels based on their corresponding filter weight norm and get speed-ups of more than 30% with a final validation accuracy close to that of the unpruned model. Another stuctured pruning algorithm analyzes the correlation between feature maps for convolutionals layer (activation values for neurons), and removes redundant components (Xavier Suau 2019).

Both unstructured and structured pruning methods are often *iterative*, which means that they require many expensive prune-retrain cycles and are thus very specific to certain architectures and tasks. Besides being computationally intensive, iterative pruning also suffers from the fact that it is highly dubious which units should be removed during a pruning iteration. Bartoldson et al. 2019 show that, contrarily to common belief, iteratively pruning weights (or filters and neurons when structured pruning is done) with a high magnitude performs just as well as pruning small magnitude weights. *One-shot* pruning methods only prune the network once. In some cases some retraining is required, after which the pruned model is obtained. If one believes that the advantages of over-parameterized neural networks come from better exploration of the feature space, or better arranged sub-networks, then the ultimate goal of pruning would be to identify "trainable" components of a network at initialization (i.e. a winning ticket) without having to train the network. The one-shot SNIP method (Lee, Ajanthan, and Torr 2018), that evaluates a connection's saliency before training and only keeps important connections, obtains extremely sparse networks with virtually the same accuracy as the reference model.

3 Results

This project focuses on classification tasks on natural images datasets, solved by neural networks. It takes a weight-vector perspective when reasoning about the sources of redundancy and the effects of pruning. This means that only structured pruning will be performed, at the level of neurons or filters. Those will also be the main entities studied when trying to understand the properties of neural network optimization. Having discussed the drawbacks of unstructured pruning, the initial idea was to reproduce the findings of the Lottery Ticket Hypothesis in a structured manner, but research suggests that the dynamics of "winning tickets", made of individual weights, do not apply to entire neurons or filters (Liu et al. 2018).

Still, extending the Lottery Ticket Hypothesis to units (neurons or convolutional filters), we would prune based on magnitude of the associated weight vectors. This is still a popular strategy when it comes to structured pruning, and we would like to investigate more theoretically sound alternatives. For that purpose, we study the sources of redundancy in neural networks, that must exist since the maximum capacity of the network is not reached in most cases, and see how they apply at the level of units (and their corresponding weight vectors). We do this in order to formulate suggestions on how we could exploit these sources

of redundancy to design new structured pruning strategies that yield efficient models with a test accuracy close to that of a reference model. The observations and discoveries made in this project can be summed up by three distinct contributions:

- Evaluate the efficiency of a new pruning strategy, that relies on angle instead of magnitude.
- Provide a visualization tool, to explore training dynamics from a weight vector perspective, where similarity is defined in terms of the angle between two weight vectors.
- Perform a subspace analysis of chosen layer weight matrices. This is the most promising and direct way of analyzing redundancy at the level of units.

The code is available at `https://github.com/jheitmann/Neural-net-pruning` . Before diving into the first part, here are the specifications of the models we used in our experiments:

Models

Network	LeNet	Conv-2	Conv-6	VGG19BN
Convolutions		64, 64, pool	64, 64, pool 128, 128, pool 256, 256, pool	2x64 pool 2x128 pool 4x256 pool 4x512 pool 4x512 pool
FC layers	300, 100, 10	256, 256, 10	256, 256, 10	avg-pool 512, 512, 10
All/Conv Weights	266K	4.3M / 38K	1.7M / 1.1M	20.0M
Iterations/Batch	50K / 64	20K / 64	40K / 60	112K / 64
Optimizer	SGD 3e-3 Momentum 0.9	Adam 2e-4	Adam 3e-4	SGD 0.1-0.01-0.001 Momentum 0.9

Figure 1: Architectures tested in this project. Convolutions are 3x3. Lenet is from LeCun, Bottou, et al. 1998. Conv-2/4/6 are variants of VGG (Simonyan and Zisserman 2014). Last model is VGG19 with batch normalization. Initializations are Gaussian Glorot (Glorot and Bengio 2010).

3.1 Comparison of pruning criteria

Starting point

As we have seen, most modern pruning methods evaluate a component's importance based on its magnitude or ℓ_2-norm (Li et al. 2016). Even though this criterion delivers satisfying results, there isn't much theory to back it up, except for the greater contribution to the next layer's input or the final output, when averaged over all possible inputs. But this is not necessarily true for the input space, Nagarajan and Kolter 2019 show that the spectral norm of the weight matrices can be significantly reduced after training, without affecting the validation accuracy too much. This so-called "noisy component" that is removed seems to have a bad alignment with the data and hence doesn't interfere too much with the parts that are more relevant to the output. Based on the findings of Brutzkus and Globerson 2019, we would like to investigate a new criterion to measure a weight vector's importance

that, instead of looking at magnitude, measures the "closeness" or "similarity" to other weight vectors. The idea that motivates this criterion is that, if indeed weight vectors get clustered during the training process, there might be some outliers that don't fall into any cluster and could be removed because they align poorly with the input data.

We would like to see if a pruning criterion that doesn't take magnitude into account can deliver satisfying results, so before working with a weight matrix $\mathbf{W}$, we normalize its rows and obtain $\hat{\mathbf{W}}$. To measure how "clustered" the weight vectors are at a given time in training, we use a metric called the *frame potential* (FP), which is defined as follows for a normalized matrix $\hat{\mathbf{W}}$:

$$\mathrm{FP}(\hat{\mathbf{W}}) = \sum_{i,j} |\langle \hat{\mathbf{w}}_i, \hat{\mathbf{w}}_j \rangle|^2,$$

where $\mathbf{w}_i$ are the normalized rows of $\hat{\mathbf{W}}$. Note that if we define the correlation matrix $\mathbf{X} = \hat{\mathbf{W}}\hat{\mathbf{W}}^T$, then this can be rewritten as

$$\mathrm{FP}(\hat{\mathbf{W}}) = \|\mathbf{X}\|_F^2 = \mathrm{tr}(\mathbf{X}^T\mathbf{X}).$$

Problem setup

Consider a dense feed-forward neural network $f(x;\theta)$ with initial parameters $\theta = \theta_0 \sim \mathcal{D}_\theta$, corresponding to the weights of N hidden layers, and one output layer. When optimizing with Stochastic Gradient Descent (SGD) on a training set, f reaches minimum validation loss l at iteration j with test accuracy a. The network has arrived at parameters θ_j. Let $\mathbf{W}^L \in \mathbb{R}^{m\times n}$ be the weight matrix associated to the L^{th} layer (hidden or output), at iteration j. Out of the m units, we would like to remove $p < m$ of them such that, given the newly obtained test accuracy a_p when dropping these units, the difference in accuracy $\Delta = a - a_p$ is minimized. Due to the combinatorial nature of the problem, we focus on greedy methods that iteratively remove units based on a certain criterion.

Magnitude pruning: Sort the weight vectors by their norm $\|\mathbf{w}_i\|, 1 \le i \le m$. Remove the weight vector with the smallest norm. Repeat the procedure p times, updating the weight matrix accordingly.

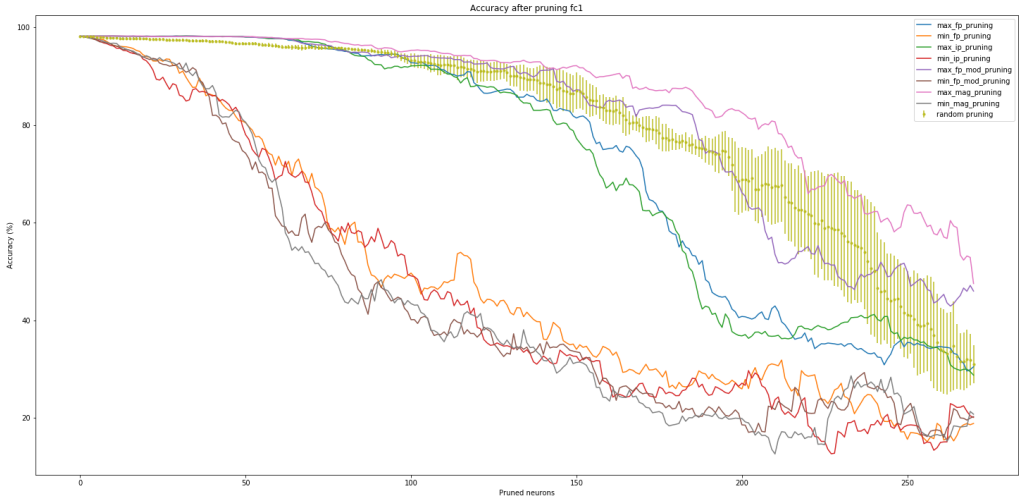
Frame-potential pruning: Sort the weight vectors by their contribution to the frame-potential, which for a weight vector $\mathbf{w}_i, 1 \le i \le m$, is $\sum_{j=1}^{m} |\langle \hat{\mathbf{w}}_i, \hat{\mathbf{w}}_j \rangle|^2$, where $\hat{\mathbf{w}}_i, \hat{\mathbf{w}}_j$ are the normalized vectors. Remove the weight vector with the smallest contribution to the frame-potential. Repeat the procedure p times, updating the weight matrix accordingly.

For the sake of comparison, we might want to remove $p < m$ units such that the difference in accuracy $\Delta = a - a_p$ is maximized, to study how much harm can be done to the trained network. Again, using magnitude or contribution to frame-potential as a metric to measure a unit's importance, we can adapt *Magnitude pruning* and *Frame-potential pruning* such that the weight vector with the highest norm or the highest contribution to the frame-potential is removed, and repeat this procedure multiple times.

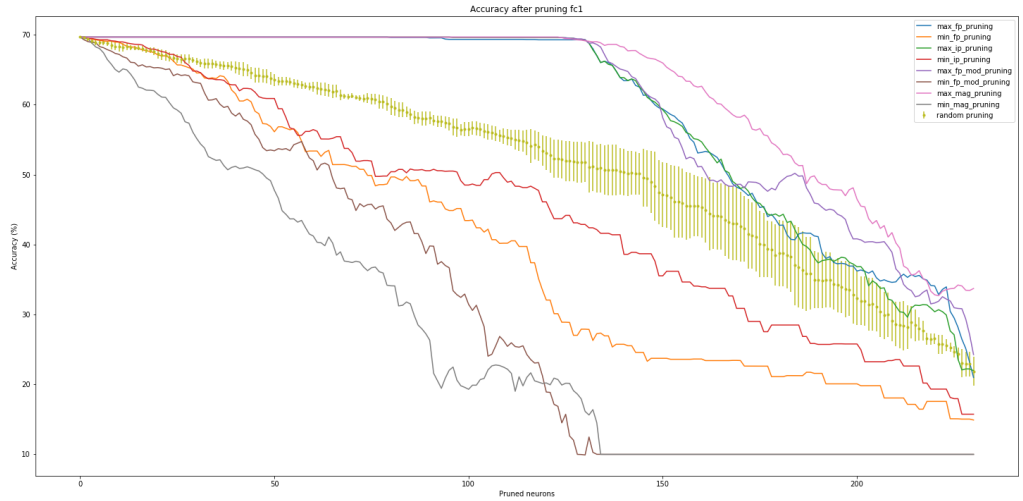
The effects of pruning without retraining were only studied for the smaller models LeNet and Conv-2. Random pruning, which removes individual units at random, has been added as a baseline. Other pruning metrics, that were not discussed in this report, have also been tested and should be disregarded. The results are summarized in figures 2a and 2b.

Observations

Magnitude pruning performs better than *Frame-potential pruning*, for both fully connected and convolutional layers (plots for convolutional layers not shown, they tell a similar story). Nonetheless, pruning strategies that maximize or minimize frame-potential of the pruned matrix show that angle with respect to other weight vectors is correlated with a unit's importance, which is measured through the observed difference in test accuracy. `max_fp_pruning` performs better than random pruning (much better in some cases as fig. 2b shows), and `min_fp_pruning` can significantly damage a network. This leads us to the next section, at this point it seemed necessary to further investigate the relationship between angle and magnitude of a unit, and a visualization that encodes both angle and magnitude allows us to better understand the dynamics between the two.



(a) Pruning the first fully connected layer of LeNet



(b) Pruning the first fully connected layer of Conv-2

Figure 2: Comparison of different pruning metrics. `max_mag_pruning` (`min_mag_pruning`) corresponds to *Magnitude pruning* that maximizes (minimizes) the Frobenius norm of the pruned matrix. `max_fp_pruning` (`min_fp_pruning`) corresponds to *Frame-potential pruning*, it maximizes (minimizes) the frame-potential of the pruned matrix. For random pruning, averages were taken over 5 random experiments. Error bars were added with a width equal to the standard deviation, computed over the 5 random experiments, of the test accuracy obtained after pruning i units.

3.2 Visualizing training dynamics

Method

For a given weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, we would like to embed the norms of the rows of $\mathbf{W}$, $\|\mathbf{w}_1\|, \ldots, \|\mathbf{w}_m\|$, and the cosine similarities between the rows in a visualization. The cosine similarities can be obtained from the correlation matrix $\mathbf{X} = \hat{\mathbf{W}}\hat{\mathbf{W}}^T$, where $\hat{\mathbf{W}}$ is the matrix we get when normalizing the rows of $\mathbf{W}$. A direct visualization of the weight vectors in an n-dimensional space is impractical, as n can be very high (e. g. $n = 16384$ for the first fully connected layer of Conv-2).

Since we mainly care about magnitude and angle, the former being a feature of a weight vector, and the latter a property between weight vectors, an undirected weighted graph seems to be an appropriate choice of visualization. More specifically, we can construct a time-varying graph, with nodes representing individual weight vectors (single units), and weights between nodes (edges) representing the closeness between weight vectors as a function of the cosine similarity. Because edge weights must be positive and the entries of the correlation matrix $\mathbf{X}$ take values in $[-1, 1]$, we first transform the cosine similarity matrix to a cosine distance matrix, i. e. we compute $\mathbf{1} - \mathbf{X}$, whose entries lie in $[0, 2]$ (0 if colinear, 1 if orthogonal, 2 if opposed direction). Next we use a Gaussian function in order to turn these cosine distances into weights:

$$W(\mathbf{w}_i, \mathbf{w}_j) = \exp(-\frac{\text{dist}^2(\mathbf{w}_i, \mathbf{w}_j)}{\sigma^2}),$$

where $\text{dist}^2(\mathbf{w}_i, \mathbf{w}_j) = (\mathbf{1} - \mathbf{X})_{ij}$ and σ is a parameter which controls the width of the Gaussian. We have obtained an adjacency matrix which we can sparsify by removing the values below a certain threshold.

To summarize, we have a time-varying graph with nodes that can increase in size (corresponding to an increase in magnitude of the weight vector) and edge weights that can increase or decrease (reflecting a higher or smaller cosine similarity). The time-steps of the graph are training epochs, thus we observe how the weight vectors in a layer of a network evolve as the network gets trained.

To visualize the graph, we apply a Force Layout that models the nodes as charges and edges as links keeping them together. In addition to a repulsive charge force, a pseudo-gravity force keeps nodes centered in the visible area and avoids expulsion of disconnected subgraphs, while links are fixed-distance geometric constraints.

Observations

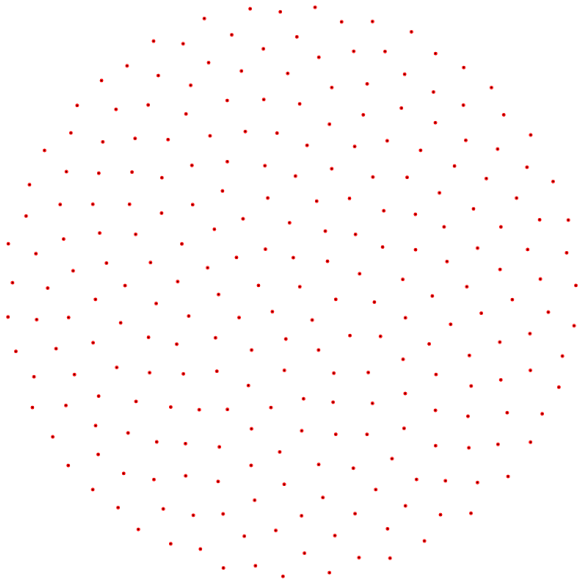
Based on the self-regularization mechanism discussed in section 2 and the results of the pruning experiments in the previous section, some basic assumptions could be made on what the visualization would show. Martin and Mahoney 2018 and Nagarajan and Kolter 2019 argue that after convergence, only a low-rank subspace of the space spanned by the rows of $\mathbf{W}$ is relevant to the output of the network. This low-complexity component is particularly receptive to training by SGD and dominates the output of the network, compared to the noisy component. This phenomenon would push weight vectors closer together and

we would observe the emergence of edges with stronger weights. Moreover, using the results of Brutzkus and Globerson 2019 and the fact that *Frame-potential pruning* has proven to work better than random pruning, we would maybe expect to see the formation of clusters, but in particular outliers that don't fall into a cluster or aren't close to any other weight vector in the weight matrix.

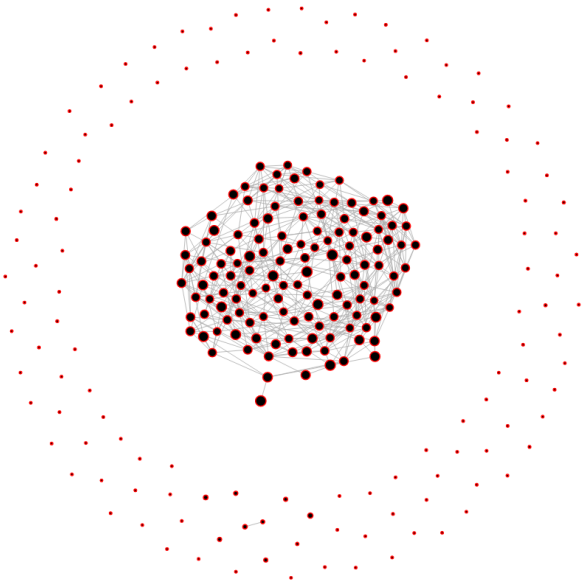
The observations confirm our initial assumptions. In general, weight vectors move closer to each other, and often we see one big cluster, with some outliers that are not part of this big connected component (see fig. 3b). The emergence of the cluster probably indicates the presence of a low-rank subspace, that corresponds to the low-complexity component. Figure 3b, showing the graph we get for the first fully connected layer of Conv-2, is particularly interesting, because it highlights multiple phenomena discussed in section 2. First, the aforementioned main big cluster, and the outliers around it, that in addition to not being part of the cluster are not close either to other weight vectors. Second, the emergence of removable units, which are units that do not experience a significant change during the training process and can be dropped out of the network without hurting the validation accuracy much. Note that in this case, since these removable units are not affected much by training, they are also not pushed towards the main cluster (most likely the low-rank subspace). This explains why in our pruning experiment for the same layer (fig. 2b), *Magnitude pruning* and *Frame-potential pruning* perform equally well, up to quite a high value of pruned units. In this particular example it becomes clear why magnitude is a good estimate of a unit's importance.

A surprising result was that in some cases, even for larger networks, the weight vectors actually get clustered, as figure 4 shows. The threshold below which edges are removed was set to a value close to 1 for this graph. This means that if there is an edge between two nodes, the corresponding weight vectors almost overlap. The distinct clusters are clearly visible. The question was whether, despite the bias term which is not represented here and the non-linearity of the activation function, overlapping weight vectors would also produce similar outputs. An alternative way of obtaining a visualization based on similarities was to compute the correlations between the outputs of the units (after adding the bias term and applying the activation function). This yields a different correlation matrix $\mathbf{X}$, that can be transformed in a similar way as above to obtain a time-varying graph. Results are not shown here, but we observe the same clusters when visualizing the last convolutional layer of VGG19BN with this method.

The next section is motivated by the wish to examine more precisely the nature of the big cluster.



(a) At initialization



(b) After convergence

Figure 3: Visualizing the training dynamics of the first fully connected layer of Conv-2. Edges corresponding to an inner-product below 0.25 are removed.

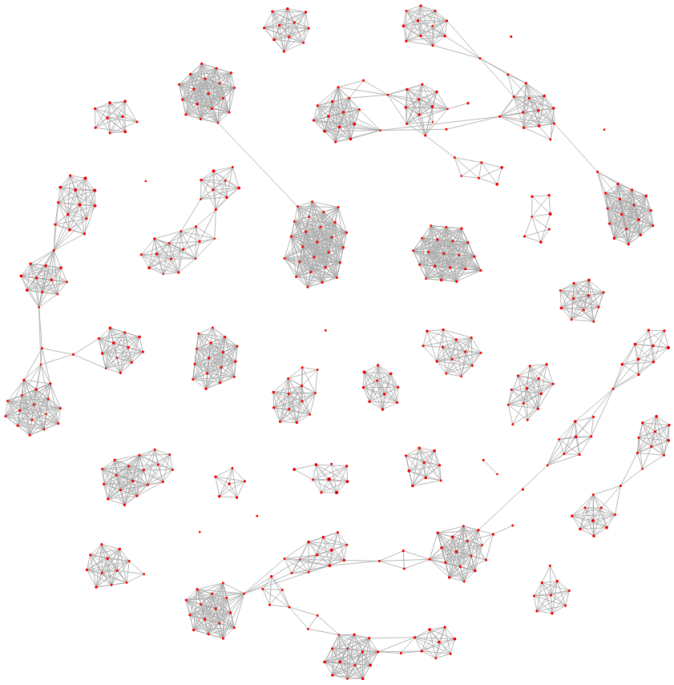


Figure 4: Cluster formation in the last convolutional layer of VGG19BN during training

Additional observations

The visualization makes it possible to compare layer weight matrices with different number of units, but same number of input connections. Let $\mathbf{W}_1 \in \mathbb{R}^{m_1 \times n}, \mathbf{W}_2 \in \mathbb{R}^{m_2 \times n}$ be two layer weight matrices. Then we can construct a new weight matrix

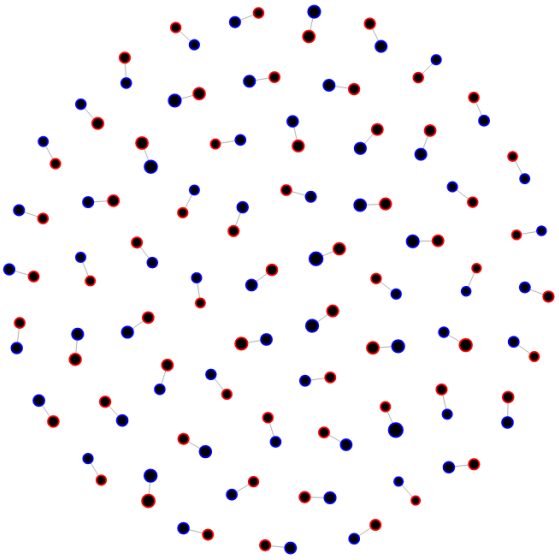
$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 \\ \\ \mathbf{W}_2 \end{bmatrix} \in \mathbb{R}^{(m_1+m_2) \times n}$$

and apply our visualization to it. This can be used to compare layers of different networks (e. g. same model but different weight values), to see if there are any correlations between the learned weight vectors of the layers.

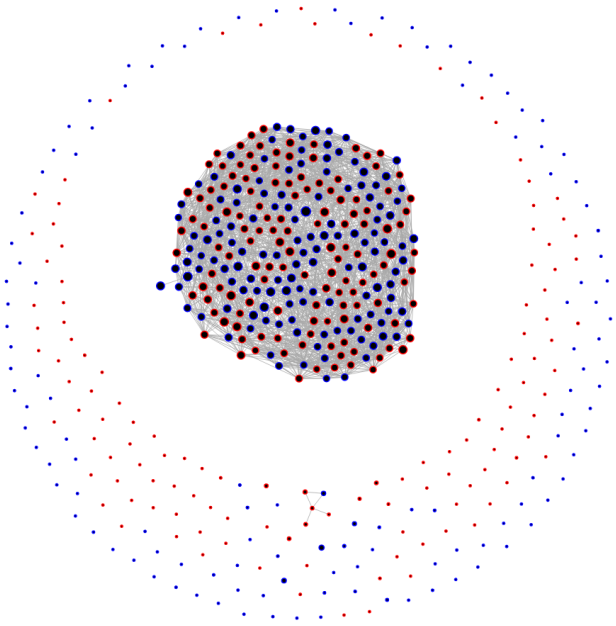
Among the experiments that were carried out with this tool, one which delivered curious results is worth mentioning in this report. The setup was the following: a Conv-2 network, which can be modeled as $f(x;\theta)$, with initial parameters $\theta = \theta_0 \sim \mathcal{D}_\theta$, was trained with SGD until it reached maximum test accuracy at iteration j, arriving at parameters θ_j. Then, the same network was reset to its initial parameters θ_0, and the weights of the classifier were re-sampled according to $\mathcal{D}_\theta$. The classifier is a component that corresponds to the 3 fully connected layers of the network (after the convolutional layers), and is present in all the VGG-variants. Thus we get a model $f(x;\theta_0')$, where θ_0' is the set of parameters that is

identical to θ_0 except for the classifier component. We then train the model until iteration j, arriving at parameters θ'_j. The question was how different the weight vectors learned in the two copies of the network would be, which lead us to compare the weight matrices of the final convolutional layer (started with identical weights, fig. 5a shows similarity after training), and the weight matrices of the first fully connected layer (started with different weights, fig. 5b shows similarity after training).

To our surprise, the two networks seem to have converged to similar solutions in the convolutional layers, as illustrated by the similarity between the weight vectors and their initial copy (fig. 5a). Also, the big cluster we observe after training in the first fully connected layer contains trained units of both models (fig. 5b). If indeed the main cluster is indication for the existence of a low-rank subspace, then it seems like the two learned subspaces are very similar (later, the principal angle between low-rank approximations was measured and confirms this). This would mean that similar importance is given to the outputs of the last convolutional layer, which is more evidence to support that the convolutional layer of the two models learned similar representations. If this turns out to be more than just a lucky observation, but rather a property that generalizes to other layers and networks, we would have to question the validity of certain interpretations of the Lottery Ticket Hypothesis. We are saying that, based on our observations, the learned representations of a layer depend only on its initialization, and maybe on the initializations of the preceding layers, but not on the initialization of the subsequent layers. This would question the existence of well-initialized subnetworks that "win the lottery", since a subnetwork that contains weights of a certain layer also includes weights of subsequent layers (in our case the classifier). A more thorough analysis should be done in the future do get more precise results.



(a) Last convolutional layer, inner-product cutoff: 0.48. All nodes are only connected to their image in the other model, which was identical to it at initialization.



(b) First fully connected layer, inner-product cutoff: 0.25

Figure 5: Model comparison after training, when re-sampling the classifier. Red nodes correspond to weight vectors of the initial model, blue nodes to weight vectors of the model with re-sampled classifier.

3.3 Subspace analysis

In light of the findings of the previous sections and based on results in model compression that make use of matrix factorization (Lebedev et al. 2014) to compress deep neural networks, we propose a subspace analysis of chosen layer weight matrices, as a way to confirm the intuition we got from the visualization, and to learn more about the properties of models of different sizes. Inspired from the use of Random Matrix Theory of Martin and Mahoney 2018 to study implicit self-regularization of neural networks, we analyze the Empirical Spectral Density (ESD) of layer correlation matrices. We also look at how performing a low-rank approximation of certain layer weight matrices affects the test accuracy, to see how informative the obtained low-complexity component is, depending on the model.

Random Matrix Theory

For a given weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, the ESD of the correlation matrix $\mathbf{X} = \mathbf{W}\mathbf{W}^T, \mathbf{X} \in \mathbb{R}^{m \times m}$ is the density of eigenvalues $\rho(\lambda)$. If $\mathbf{Z}$ denotes a $m \times n$ random matrix whose entries are independent identically distributed random variables with mean 0 and variance $\sigma^2 < \infty$, let

$$\mathbf{Y}_n = \frac{1}{n}\mathbf{Z}\mathbf{Z}^T, \mathbf{Y}_n \in \mathbb{R}^{m \times m}$$

Then Marchenko-Pastur (MP) theory states that the ESD of $\mathbf{Y}_n$ has the limity density given by the MP distribution $\rho(\lambda)$. Note that $\rho(\lambda)$ is parametrized by the variance σ^2 and the aspect ratio $Q = n/m$ of the matrix $\mathbf{Z}$. In our case, $(\mathbf{W})_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2)$, where σ^2 is determined by the Gaussian Glorot distribution, and

$$\begin{aligned} \mathbf{X} &= \mathbf{W}\mathbf{W}^T \\ &= \frac{1}{n}(\sqrt{n}\mathbf{W})(\sqrt{n}\mathbf{W}^T) \end{aligned}$$

Since $(\sqrt{n}\mathbf{W})_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, n\sigma^2)$, the limiting distribution of the ESD of $\mathbf{X}$ depends only on $Q = n/m$ and $n\sigma^2$.

Orthogonal Projections

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a real (possibly rectangular) matrix with $m \geq n$. Suppose that $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the Singular Value Decomposition (SVD) of $\mathbf{A}$. Recall that $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, and $\mathbf{\Sigma}$ is an $m \times n$ diagonal matrix with entries $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. Then the Eckart–Young–Mirsky theorem states that the best rank k approximation to $\mathbf{A}$ in the spectral norm, denoted by $\|\cdot\|_2$, is given by

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

where $\mathbf{u}_i$ and $\mathbf{v}_i^T$ denote the i^{th} column of $\mathbf{U}$ and $\mathbf{V}$ respectively.

We are interested in the row-space of a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, to study how implicit self-regularization mechanism operate at the level of units. Thus we perform a rank k approximation of the matrix $\mathbf{W}^T, k < m$, denoted by $\mathbf{W}_k^T$, before replacing $\mathbf{W}$ by $\mathbf{W}_k$ in the original network. To find a good value of k, the elbow method is applied to the function that maps the rank $k, 1 \leq k < m$, to the residuals obtained when performing a low-rank approximation of rank k.

Observations

Figures 6, 7b and 7b show the empirical spectral densities we get for different layers. Without adding an explicit regularization term when optimizing with SGD, all layer weight matrices converge to a low-complexity solution (with a small nuclear norm), as demonstrated by the large number of eigenvalues close to 0 and just a few bigger eigenvalues. This seems to be particularly true for larger models such as VGG19BN (fig. 7b and fig. 7b), where there is no tail-off close to 0 in the distribution of eigenvalues but rather a drop.

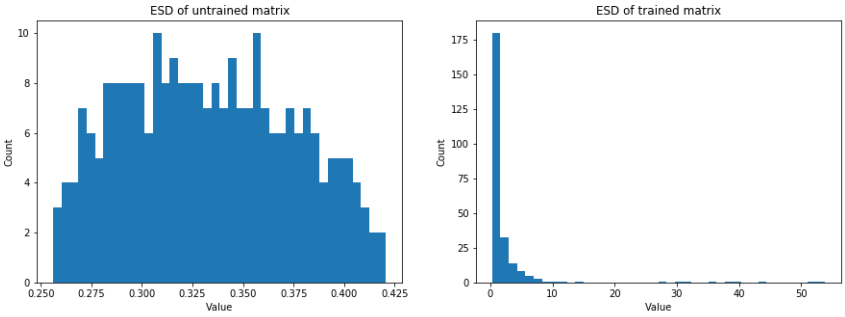
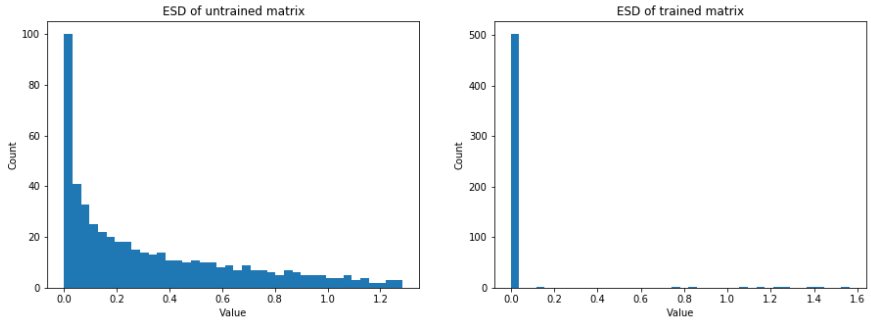
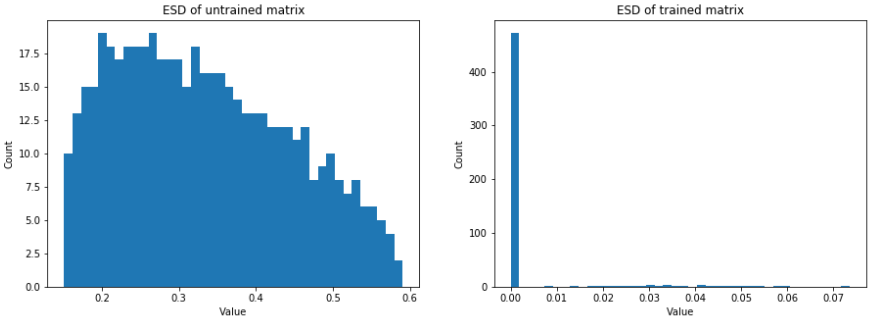


Figure 6: Empirical spectral densities of Conv-2 first fully connected layer correlation matrix, $Q = 64$



(a) First fully connected layer, $Q = 1$



(b) Last convolutional layer, $Q = 9$

Figure 7: Empirical spectral densities of VGG19BN layer correlation matrices

Network	Conv-2	Conv-6	VGG19BN
Number of neurons	256	256	512
Aspect ratio Q	64	16	1
Rank k	10	10	10
$100 \cdot \frac{\|\mathbf{W}_k\|_F}{\|\mathbf{W}\|_F}$	70.25 ($s^2 = 1.39$)	61.43 ($s^2 = 1.27$)	99.99
$a - a_{\text{proj}}$	6.34 ($s^2 = 1.29$)	1.61 ($s^2 = 1.16$)	0

Figure 8: Low-rank approximation of first fully connected layer, tested with different architectures. For Conv-2 and Conv-6, values are averages taken over 10 random experiments. a_{proj} denotes the accuracy obtained after replacing $\mathbf{W}$ by $\mathbf{W}_k$.

Figure 8 confirms this observation, when performing a rank 10 approximation of the first fully connected layer's weight matrix, there is no drop in test accuracy for the VGG19BN

model, and the norm of the weight matrix is almost unaffected by the approximation. We observe that the deeper the model, the smaller the drop in test accuracy after the rank 10 approximation. Note that with increasing depth, the aspect ratio Q of the first fully connected layer's weight matrix decreases. A smaller value of Q might result in a better exploration of the feature space, which in turn might help to find more robust solutions, that are less complex. Finally, we see that although the first fully connected layer of Conv-6 experiences a higher drop in Frobenius-norm after a low-rank approximation than the one of Conv-2, the drop in test accuracy is smaller. A possible explanation could be that in deeper networks, the noise component of a weight matrix has worse alignment with the outputs of the last convolutional layer, hence has less of an impact on the output of the network. As for VGG19BN, the noise component was almost entirely suppressed over the course of training. One should keep in mind that VGG19BN was trained for longer, compared to the other networks, and with a learning rate schedule, which allowed to squeeze out very fine-scale structures.

On a side node, it is curious that irrespective of the model, the elbow value for the rank k of the approximation is 10 for all first fully connected layers. 10 is also the number of classes in the CIFAR-10 dataset, and it would be interesting to check whether this also applies to a different number of classes.

4 Discussion

We have seen that indeed sources of redundancy emerge at the level of units of a neural network, observing for instance removable neurons in the first fully connected layer of Conv-2, or repeated (clustered) filters in the last convolutional layer of VGG19BN. A naive pruning strategy that removes those types of units suffers from the fact that first, removable units are not that frequent in larger networks and second, it is unclear how to properly remove redundant units. If we want to prune more heavily, we need to exploit other sources of redundancy.

The problem with iterative methods that prune multiple times during training is that we alter the already complicated geometry of the loss function, in a way that cannot be foreseen. Most iterative methods lack a theoretical explanation, and fail to deliver results that could not have been reached by smaller networks.

Some weight matrices (if not all) of a neural network are undoubtedly low-rank after training, as the empirical spectral densities of VGG19BN layer correlation matrices show. We think that getting a good understanding of this low-rankness might help us in designing better pruning strategies.

Finally, we think that this project raised some questions that are interesting from a theoretical point of view: Why do the weight vectors of some layers perfectly cluster around just a few directions, whereas in other layers this doesn't happen? Are the weight vectors in a layer learned more or less independently of the subsequent layers?

Hopefully these questions can be investigated in future work.

References

Bartoldson, Brian R et al. (2019). "The Generalization-Stability Tradeoff in Neural Network Pruning". In: *arXiv preprint arXiv:1906.03728*.

Brutzkus, Alon and Amir Globerson (2019). "Why do Larger Models Generalize Better? A Theoretical Perspective via the XOR Problem". In: *International Conference on Machine Learning*, pp. 822–830.

Casper, Stephen et al. (2019). "Removable and/or Repeated Units Emerge in Overparametrized Deep Neural Networks". In: *arXiv preprint arXiv:1912.04783*.

Frankle, Jonathan and Michael Carbin (2018). "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635*.

Frankle, Jonathan, Gintare Karolina Dziugaite, et al. (2019). "The Lottery Ticket Hypothesis at Scale". In: *arXiv preprint arXiv:1903.01611*.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.

Goldt, Sebastian et al. (2019). "Dynamics of stochastic gradient descent for two-layer neural networks in the teacher-student setup". In: *arXiv preprint arXiv:1906.08632*.

Han, Song et al. (2015). "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*, pp. 1135–1143.

Hassibi, Babak and David G Stork (1993). "Second order derivatives for network pruning: Optimal brain surgeon". In: *Advances in neural information processing systems*, pp. 164–171.

Lebedev, Vadim et al. (2014). "Speeding-up convolutional neural networks using fine-tuned cp-decomposition". In: *arXiv preprint arXiv:1412.6553*.

LeCun, Yann, Léon Bottou, et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann, John S Denker, and Sara A Solla (1990). "Optimal brain damage". In: *Advances in neural information processing systems*, pp. 598–605.

Lee, Namhoon, Thalaiyasingam Ajanthan, and Philip H. S. Torr (2018). "SNIP: Single-shot Network Pruning based on Connection Sensitivity". In: *arXiv preprint arXiv:1810.02340*.

Li, Hao et al. (2016). "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710*.

Liu, Zhuang et al. (2018). "Rethinking the value of network pruning". In: *arXiv preprint arXiv:1810.05270*.

Martin, Charles H and Michael W Mahoney (2018). "Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning". In: *arXiv preprint arXiv:1810.01075*.

Morcos, Ari S et al. (2019). "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers". In: *arXiv preprint arXiv:1906.02773*.

Nagarajan, Vaishnavh and J Zico Kolter (2019). "Uniform convergence may be unable to explain generalization in deep learning". In: *Advances in Neural Information Processing Systems*, pp. 11611–11622.

Neyshabur, Behnam et al. (2018). "Towards understanding the role of over-parametrization in generalization of neural networks". In: *arXiv preprint arXiv:1805.12076*.

Rolnick, David et al. (2017). "Deep learning is robust to massive label noise". In: *arXiv preprint arXiv:1705.10694*.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Tian, Yuandong et al. (2019). "Luck Matters: Understanding Training Dynamics of Deep ReLU Networks". In: *arXiv preprint arXiv:1905.13405*.

Xavier Suau Luca Zappella, Nicholas Apostoloff (2019). "Filter Distillation for Network Compression". In: *arXiv preprint arXiv:1807.10585*.

Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide residual networks". In: *arXiv preprint arXiv:1605.07146*.

Zhang, Chiyuan et al. (2016). "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530*.