Data Science Semester Project

Author: Julien Heitmann

Sources of Redundancy in Neural Networks

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-parametrised networks generalise poorly without further regularisation, experiments show that in most cases, increasing the number of parameters does not worsen the capability to generalise. 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 generalise better. This paradox is one of the big unresolved questions of deep learning, and understanding the phenomena behind it would deliver a lot of insights about why neural networks work so well. 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 pruning. The goal is to identify both similarities and differences in the theories, and provide evidence in favor or against them in a series of experiments. Moreover, some visualization tools will be introduced that will help getting a better intuition for the complex process that is neural network training.

1.1 Formal background

Formally define frame potential, generalization error, network architecture, optimization algorithms.

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 (the residuals) that is removed when projecting onto the top singular vectors 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 initialisation, which leads to the formation of subnetworks that can be trained efficiently. The second one states that overparametrisation 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 generalisation. 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 aranged 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 speciliased 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-initialised. 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 initialise 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 initialisations generalise 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.

Coming back to the second question: XOR-problem, sparse subspace spanned by trained vectors. Question arises: will optimization algorithms such as SGD enforce this alignment, therefore preventing overfitting? Interplay between vector alignment and "trainable" vectors. Goal is to explore and go beyond, understanding this mechanism will help us design better optimization algorithms, pruning methods, architectures, initialization methods, etc. Collapse at last layer?

Additional work

Better exploration, luck and overlap, alignment of noise subspace.

2.3 Leveraging model simplicity: Compression and Pruning

Structured vs unstructured pruning, surprisingly leads to better generalization, pruning as a noise signal. Speed-up, memory foot-print. Need over-parameterized network for better exploration of hidden layer space (give examples), once the model converges / during training, remove useless components. Surprising results: prune components with highest magnitude, can lead to better test accuracy after re-training the pruned network.

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. Why not look at unstructured

pruning / individual weights?

The fact that the lottery ticket hypothesis might not really apply at the level of nodes / filters (Liu et al. 2018) could be due to multiple reasons:

- It is more difficult to identify winning tickets at the level of nodes or filters, i. e. iterative pruning based on the ℓ_2-norm fails to identify elements that are part of the winning ticket
- Structured pruning is to restrictive. Unstructured pruning can yield very complicated architectures that have at least one weight in every node or filter, even at high pruning ratios. This does not apply to structured pruning.
- Structured pruning is only efficient at low pruning rates, where the pruned architecture might be able to reach the same training loss when randomly initialized. Thus pruning can be done at the beginning, and there is no need to identify a winning ticket.

It would be interesting, from a theoretical point of view, to find out if, similarly to the well-initialized weights in the standard setting of the lottery ticket hypothesis, "trainable" nodes or filters exist. The hypothesis doesn't directly support this as, at a given layer, every node or filter should have the same expected number of weights which are in the winning ticket. But does that imply that, on average, at a given layer all nodes or filters are equally important and trained? In other words, are weights that are part of a winning ticket uniformly distributed across the nodes or filters of a layer, or do they concentrate on just a few of them?
Nonetheless, larger networks might still generalize better, and when we do not know how complex a classification task is or how big of a network we should use, then training an over-parameterized network can be interesting, especially if we are able to quickly identify "trainable" nodes or filters, so that other parameters can be pruned and training made faster.
Assuming that the lottery ticket hypothesis is true, all nodes or filters of a layer in a neural network have the same probability of being part of the "trainable" subnetwork at initialization, that is having some weights that are in the winning ticket. Also, at every layer, all nodes or filters have the same expected number of weights in the winning ticket.

Marginal distribution? Given that the node contains at least one weight that is part of the winning ticket, how does that affect the probability of other weights in the same node or filter of being part of the winning ticket?

Models

Network	LeNet	Conv-2	Conv-6	VGG19
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 et al. 1998. Conv-2/4/6 are variants of VGG (Simonyan and Zisserman 2014). 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 (**CITE**). 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 saliency 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 one input layer, 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 $\mathbf{W}^L \in \mathbb{R}^{n\times m}$ be the weight matrix associated to the L^{th} layer (input, hidden or output), at iteration j. Out of the n units, we would like to remove $p < n$ 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 $\|\mathbf{w}_i\|, 1 \leq i \leq n$. Remove the weight vector with the smallest norm. Repeat the procedure p

times, updating the weight matrix accordingly.

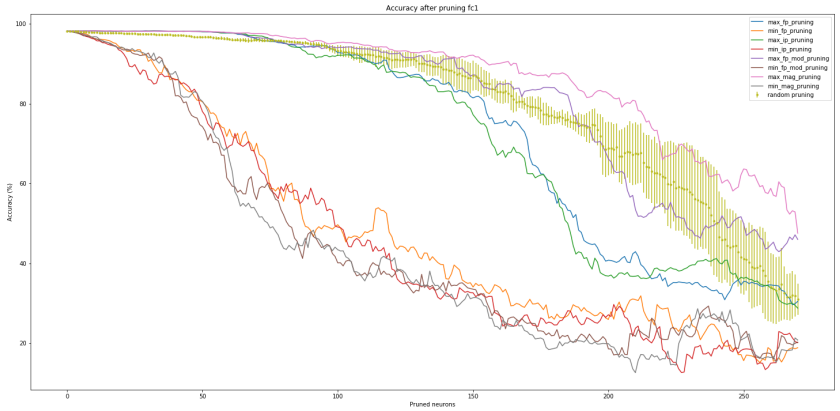
Frame-potential pruning: Sort the weight vectors by their contribution to the frame-potential, which for a weight vector $\mathbf{w}_i, 1 \leq i \leq n$, is $\sum_{j=1}^{n} |\langle \hat{\mathbf{w}}_i, \hat{\mathbf{w}}_j \rangle|^2$, where $\hat{\mathbf{w}}_i, \hat{\mathbf{w}}_j$ are the normalized vectors. 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 $p < n$ 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 saliency, 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.

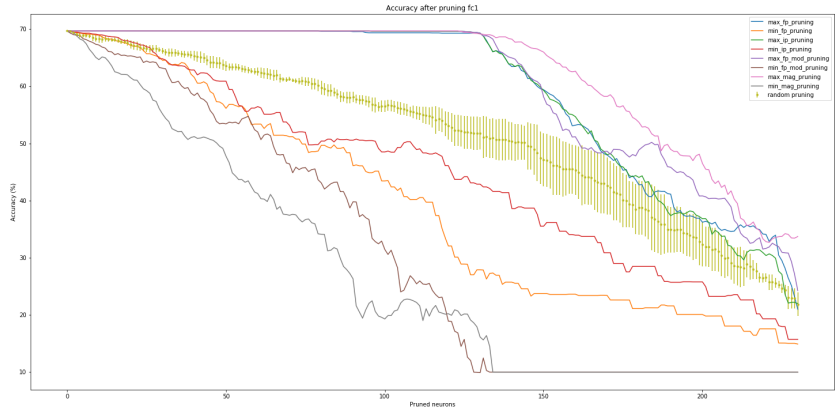
The effects of pruning without retraining were only studied for the smaller models LeNet and Conv-2. Random pruning, which removes individual units at random, has been added as a baseline. Other pruning metrics, that were not discussed in this report, have also been tested and should be disregarded. 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 $\mathbf{W} \in \mathbb{R}^{n \times m}$, we would like to embed the norms of the rows of $\mathbf{W}$, $\|\mathbf{w}_1\|, \ldots, \|\mathbf{w}_n\|$, 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 m-dimensional space is impractical, as m can be very high (e. g. $m = 16384$ 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

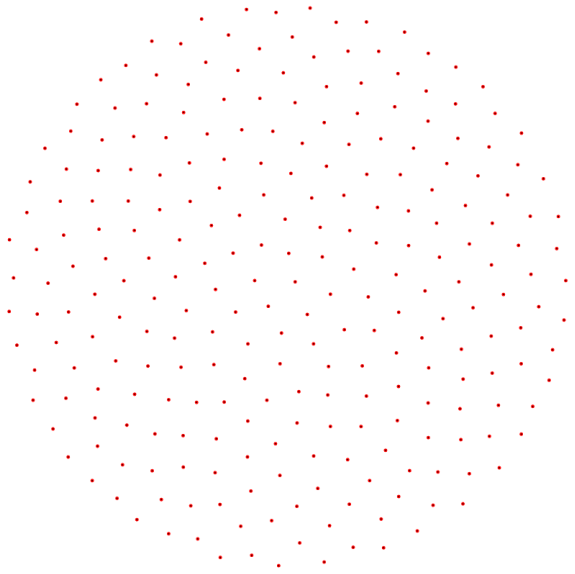
Based on the self-regularization mechanism discussed in section 2 and the results of the pruning experiments in the previous section, some basic assumptions could be made on what the visualization would show. Martin and Mahoney 2018 and Nagarajan and Kolter 2019 argue that after convergence, only a low-rank 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

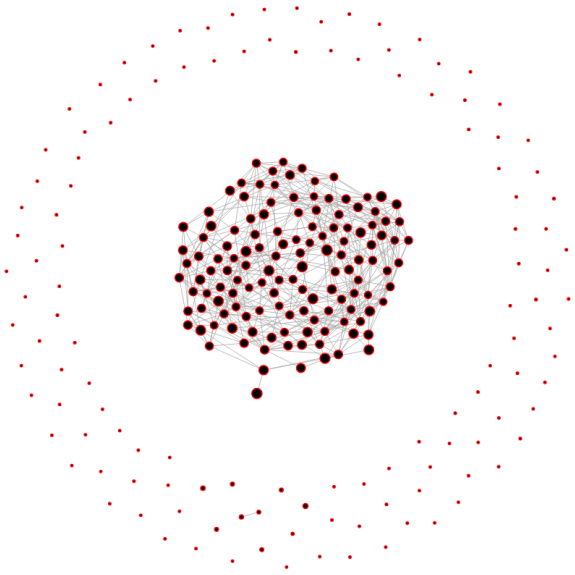
for the same layer (fig. 2b), *Magnitude pruning* and *Frame-potential pruning* perform equally well, up to quite a high value of pruned units. In this particular example it becomes clear why magnitude is a good estimate of a unit's saliency.

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. has been added as a feature to the visualization. Reveal more repeated units. Indeed repeated units, corresponding to clustered weight vectors. **TODO: add viz result.**

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

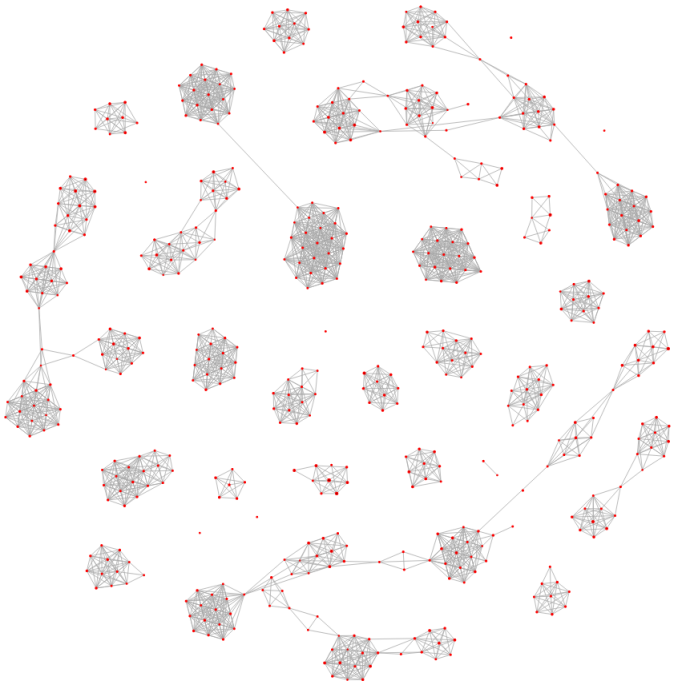


Figure 4: Cluster formation in the last convolutional layer of VGG19 during training

3.3 Subspace analysis

TODO

3.4 Additional results

TODO

4 Discussion

Write Conclusion and discuss future work. Even though emergence of removable and repeated units, implicit self-regularization operates at a different level. Subspace, study more in detail what they mean (singular vectors) in terms of the previous layer, to design better pruning methods.

References

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.

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

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*.

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*.