Evolution of Eigenvalue Decay in Deep Networks

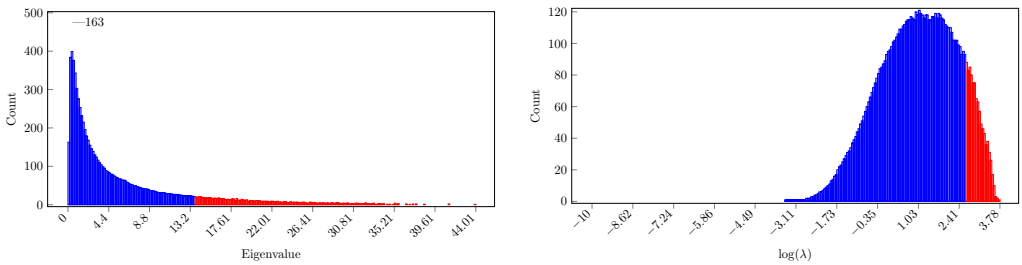
Anonymus Author
Earth

Abstract

The linear transformations in converged deep networks show fast eigenvalue decay. The distribution of eigenvalues looks like a Heavy-tail distribution, where the vast majority of eigenvalues is small, but not actually zero, and only a few spikes of large eigenvalues exist. We use a stochastic approximator to generate histograms of eigenvalues. This allows us to investigate layers with hundreds of thousands of dimensions. We show how the distributions change over the course of image net training, converging to a similar heavy-tail spectrum across all intermediate layers.

1 Motivation

The study of generalization in deep networks has shifted its focus from the skeleton structure of neural networks [2] to the properties of the linear operators of the network layers [4, 5, 13]. Measures like matrix norms, including standard Frobenius, other p-Norms or spectral norm) or stable rank [3] are important components of theoretical bounds on generalization. All of these measures rely on the singular values of the linear maps A, or equivalently on the eigenvalues of the operator times its transpose AA^T. In order to visually inspect the eigenvalue spectrum of a matrix, it is useful to compute a histogram. Histograms allows us to roughly estimate the distribution of eigenvalues and detect properties like decay behavior or the largest eigenvalues. An example of such a histogram is Figure 1, that shows the eigenvalues of a Convolution-Layer in a squeeze_net network that maps to a feature map of dimension $256 \times 13 \times 13 = 43,264$. It shows many interesting, but not well-understood characteristic properties of fully trained networks: A Heavy-tail eigenvalue distribution with the vast majority of eigenvalues being near zero, though none are actually zero, and only few spikes of large eigenvalues. Martin and Mahoney show this phenomenon in the linear layer of large pre-trained models[11], we also show it in the convolution layers and follow its evolution over the course of optimization.



(a) Linear Scale (b) Log-Scale

Figure 1: Example Histogram of a fully-trained convolution layer in a squeeze_net architecture.

Computing singular values exactly is a costly operation, particularly because the dimension in state-of-the art convolutional neural networks often exceeds 100,000. Fortunately when we are interested

Preprint. Under review.

only in a histogram, we do not need to know the exact eigenvalues, but only the number of eigenvalues that fall into the bins of the histogram.

Based on the decay property exhibited in deep networks, we propose an approach for estimating histograms of eigenvalues in deep networks based on two techniques: For estimating the few high eigenvalues, called spikes, and particularly the largest eigenvalue, we use ARPACK, a truncated eigenvalue decomposition method that does not require the matrix explicitly, but accesses it only via matrix-vector products. For estimating the remainder, called bulk, we use a method based on matrix Chebyshev approximations and Hutchinson's trace estimator [12]. Like ARPACK, it only accesses the matrix via matrix-vector products. In Figure 1, we have colored the bins we computed exactly in red, the approximated are blue.

2 Method

We denote the number of eigenvalues of a symmetric linear operator $A : \mathbb{R}^d \to \mathbb{R}^d$ that fall into the bin $[l, u)$ by $\#(A, l, u)$. We denote the eigenvalues of A by $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d$.

2.1 Implicit Matrices in Neural Networks

The highest-dimensional linear operators in deep networks used in production environments are probably convolution layers. These layers transform feature maps with number of raw-features in the input- and output feature often exceeding 100k. This is feasible because the convolution operator is not implemented as matrix-vector multiplication with dense weight matrices, but specialized and highly-optimized convolution routines are used. Really every reputable deep learning software framework provides these routines.

We can use these same routines when we estimate the eigenvalues of the linear maps of network layers. We first make sure that the network layer does not add a bias term[1]. Now let H be the linear map of a neural network layer. The forward-pass of that layer computes Hx efficiently, whereas the backward-pass computes $H^T y$ with backward-flowing gradient information y. We are interested in the eigenvalues of HH^T, hence to compute $HH^T y$, we first pass y through the backward pass of H, and pass the resulting gradient through the forward pass to obtain the resulting vector.

2.2 Estimating the Spikes

To estimate the spikes and particularly the largest eigenvalue, we use the implicitly restarted Lanczos method as implemented in the ARPACK software package. It computes a truncated eigenvalue decomposition for implicit matrices that are accessed via matrix-vector products [10]. We specify a number of spikes $T > 0$ and compute the first T eigenvalues with ARPACK. From the largest eigenvalue, we derive the equidistant histogram-binning over the range $[0, \lambda_1]$.

2.3 Estimating the Bulk

We use a technique for stochastically estimating eigenvalue counts proposed by Napoli et al.[12]. It requires that all eigenvalues fall into the range $[-1, 1]$. Hence, we first transform the matrix via $A \mapsto (2\lambda_1^{-1} A - I)$ since we already know λ_1 from the ARPACK-based spike estimator.

We define the indicator function $\delta_{l,u}(\lambda)$ that is 1 iff. $l \leq \lambda < u$ and notice that we can write the number of eigenvalues in $[l, u)$ as

$$\#(A, l, u) = \sum_{i}^{d} \delta_{l,u}(\lambda_i)$$

We can approximate $\delta_{l,u}$ with Chebyshev polynomials of a fixed degree $\delta_{l,u}(x) \approx \sum_{k=0}^{K} b_k \phi_k(x)$ where $\phi_k(x)$ is the kth Chebyshev basis and $b_k \in \mathbb{R}$ its corresponding coefficient. These coefficients are

[1] Usually deep networks apply affine transformations not linear transformations. We simply set the bias vector to zero to fix this.

Algorithm 1 Estimate Bulk(linear map A, bin $[l, u)$, degree K, numberOfSamples S)

Compute coefficients b_k for $k = 0, ..., K$ according to (1).
Sample $X = [x_1, ..., x_S]$
for $i = 1, ..., S$ **do**
  $f_2 = x_i$
  $f_1 = Af_2$ // in neural networks this is `H.forward(H.backward(`f_2`))`
  $t_s = b_0 x_s^T f_2 + b_1 x_s^T f_1$
  for $k = 2, ..., K$ **do**
    $f = 2Af_1 - f_2$ // in neural networks this is `2*H.forward(H.backward(`f_1`))`$-f_2$
    $f_2 = f_1$
    $f_1 = f$
    $t_s = t_s + b_k x_s^T f$
return $S^{-1} \sum_{s=1}^{S} t_s$

known for the indicator function[12]:

$$b_k = \begin{cases} \pi^{-1}(\arccos(l) - \arccos(u)) & k = 0 \\ 2\pi^{-1}k^{-1}(\sin(k \cdot \arccos(l)) - \sin(k \cdot \arccos(u))) & k > 0 \end{cases} \quad (1)$$

Now we can rewrite the count as the trace of a polynomial matrix function[2] applied to our matrix of interest, as it holds that $\mathrm{tr} f(A) = \sum_i f(\lambda_i)$

$$\#(A, l, u) \approx \sum_i^d \sum_k^K b_k \phi_k(\lambda_i) = \mathrm{tr} \sum_{k=0}^K b_k \Phi_k(A)$$

where $\Phi_k(A)$ is the kth Chebyshev base for matrix functions. This quantity in turn can be approximated using stochastic trace estimators, aka Hutchinson estimators [8]. It holds that $trA = \mathbb{E}_x x^T Ax$ where each component of x is drawn independently from a zero-mean distribution like standard normal or Rademacher. This expression lends itself to a simple sampling algorithm, where we draw S independent $x_1, ..., x_S$ and estimate

$$\#(A, l, u) \approx \frac{1}{S} \sum_{i=1}^{S} x_i^T \sum_k^K b_k \Phi_k(A) x_i$$

We do not have to explicitly compute $\Phi_k(A)$, as only the product $\Phi_k(A)x_i$ is required. Since Chebyshev polynomials by construction follow the recursion $\Phi_{k+1}(A) = 2A\Phi_k(A) - \Phi_{k-1}(A)$, we derive Algorithm 1 to estimate the count.

3 A Study of ImageNet Training with squeeze_nets

Our experiments are based on a pyTorch implementation of the proposed histogram estimator. We train a squeeze_net architecure [9] on imagenet data. After 30 epochs of training, we reduce the learning rate to 10%, and repeat this after another 30 epochs. We train using plain stochastic gradient descent with mini-batches of size 128 and compute histograms for all convolution layers before the first and after every epoch. For the histogram computation, we use a budget of 1000 for the exact computation of eigenvalues and approximate the remainder using the stochastic estimator.

We present some histograms in Figures 3 and 4 for the first and last convolution layers[3]. The histograms of the other layers show similar behavior as the last layer, for instance Figure 1a shows an intermediate layer after the first epoch of training quite similar to Fig. 4b, but with less extreme decay.

[2] A real-valued function $f(x) : \mathbb{R} \to \mathbb{R}$ has a corresponding matrix function $f(A) : \mathbb{R}^{m \times m} \to \mathbb{R}^{m \times m}$ and the eigenvalues of $f(A)$ are $f(\lambda_1), ..., f(\lambda_m)$. For polynomials, we get this matrix function by replacing scalar multiplications with matrix multiplications and scalar additions with matrix additions. For other classes of functions and a comprehensive introduction to matrix functions see Highham's book[7].

[3] Addional and animated histograms are available at `https://whadup.github.io/Resultate/`, however note that the website is not sufficiently anonymized for double-blind reviewing. Proceed with caution.

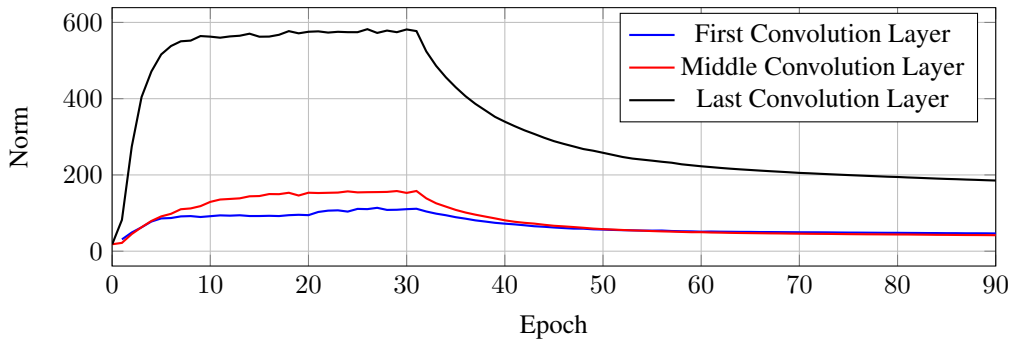


Figure 2: The norms $||\cdot||_\sigma^2$ of three layers during training

Like Martin and Mahoney [11] we identify different phases in the spectograms. Right after initialization, the matrix behaves almost as random matrix theory suggests given the element-wise independent random initialization with Gaussian random variables [14]. This can be observed in Figures 3a and 4a. However, we note that on the first layer Fig. 3a, there are some unexpected bumps in the histogram. We conjecture that this may be due to padding in the convolutions.

As optimization commences, we start to see heavytail behavior. Already after one epoch of traning, the largest eigenvalues have seperated from the bulk, while the majority of eigenvalues remains in the same order of magnitude as before training. This can be seen for the first and large convolution layer in Figures 3b and 4b. The bumps in the first layer smooth out a little.

Over the course of the first 30 epochs, the largest eigenvalues grow steadily as the tail of the spectrum grows further. Then as soon as the learning rate is reduced to 10%, the operator norms of the linear maps start to decrease as depicted in Figure 2. Considering the importance of the operator norm in known generalization bounds for feed-forward networks, this suggests that some sort of regularization is happening.

The bumps in the first layer smooth out further, but remain visible. The last layer for the most part keeps its shape in the last 60 epochs, beside the reduction of the norm we notice that the largest bar decreases in size from 4157 to 2853 and that the difference seems to move to the other bars in the blue portion of the histogram.

4 Conclusion

Understanding the structure in the linear transformations might be an important aspect of understanding generalization in deep networks. To this end we have presented a stochastic approach that allows us to estimate the eigenvalue spectrum of these transformations. We show how the spectrum evolves during imagenet training using convolutional networks, more specifically squeeze_net networks.

In the future we want to apply similar approaches to estimating the covariance structure of the intermediate feature representations and investigate the relations between covariance matrices and parameter matrices. Since the estimator we use is differentiable [6, 1], it may be interesting to investigate its usefulness for regularization.

References

[1] Ryan P. Adams, Jeffrey Pennington, Matthew J. Johnson, Jamie Smith, Yaniv Ovadia, Brian Patton, and James Saunderson. Estimating the Spectral Density of Large Implicit Matrices. Technical Report 3, 2018.

[2] Martin Anthony and Peter L Bartlett. *Neural Network Learning: Theoretical Foundations*. 1999.

[3] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. 2018.

[4] Peter Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, number 30, 2017.

[5] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-Independent Sample Complexity of Neural Networks. In Philippe Bubeck, Sebastien and Perchet, Vianney and Rigollet, editor, *Proceedings of the 31st Conference On Learning Theory*, number 1, pages 297–299. PMLR, 2018.

[6] Insu Han, Haim Avron, and Jinwoo Shin. Optimizing Spectral Sums using Randomized Chebyshev Expansions. pages 1–21, 2018.

[7] Nicholas J. Higham. *Functions of Matrices - Theory and Computation*. 2014.

[8] M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 1990.

[9] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[10] R B Lehoucq, D C Sorensen, and C Yang. *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Software, Environments, Tools. Society for Industrial and Applied Mathematics, 1998.

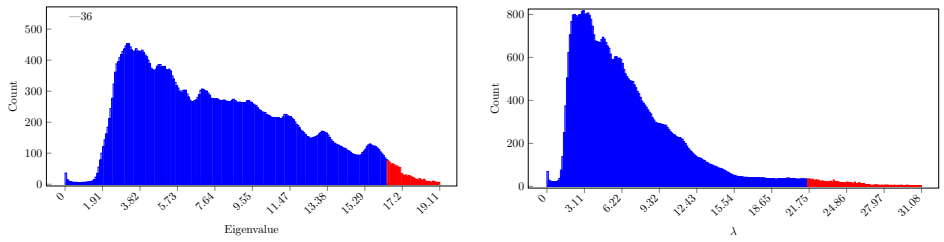
[11] Charles H. Martin and Michael W. Mahoney. Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning. pages 1–59, 2018.

[12] Edoardo Di Napoli, Eric Polizzi, and Yousef Saad. Efficient estimation of eigenvalue counts in an interval. *Numerical Linear Algebra with Applications*, 2016.

[13] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-Based Capacity Control in Neural Networks. *Proceedings of The 28th Conference on Learning Theory*, 40:1376–1401, 2015.

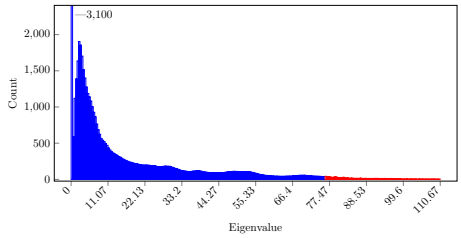
[14] Joel A. Tropp. Second-order matrix concentration inequalities. *Applied and Computational Harmonic Analysis*, (December), 2015.

Appendix

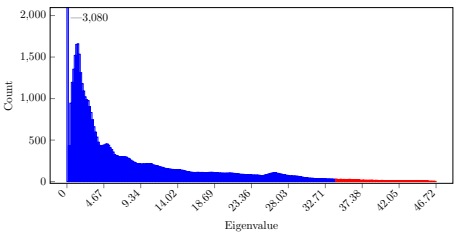


(a) Initialization

(b) Epoch 1

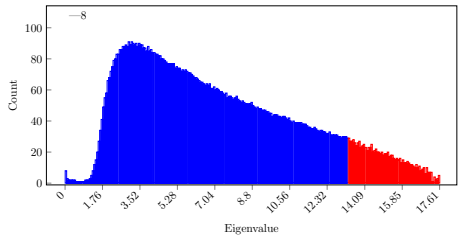


(c) Epoch 30

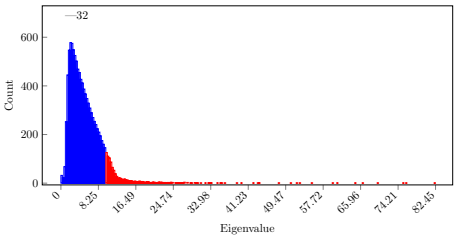


(d) Epoch 90

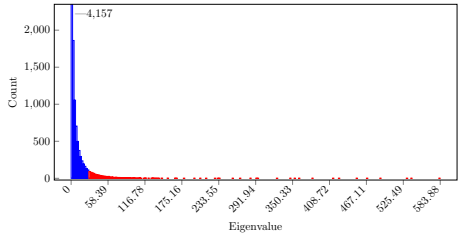
Figure 3: The first 3×3 convolution layer in the network at various stages. During all stages of optimization it shows some bumps in the histogram, that are more pronounced in earlier epochs.



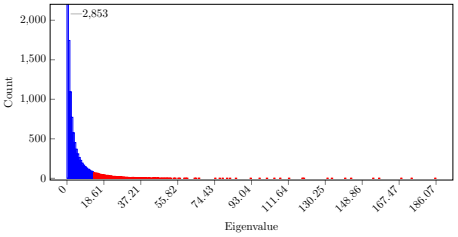
(a) Initialization



(b) Epoch 1



(c) Epoch 30



(d) Epoch 90

Figure 4: The last 3×3 convolution layer in the network at various stages. After the 30-th epoch the norm of the linear map starts to decrease and stabilized at around 190.