Approximating the Local Learning Coefficient in Neural Networks: A Comparative Analysis of Power Series Expansion Orders

Sid Baines **Ayush Bharadwaj** **George Ingebretsen** **Hernan Iriarte**

Maria Matveev

Lucius Bushnaq
(Mentor, Apollo Research)

Contents

1 Introduction **2**
1.1 Statement of Problem . . . 2
1.2 Background . . . 2
1.2.1 Power expansion of the behavioral loss . . . 2
1.2.2 Hessian . . . 3
1.3 The Hessian rank as an upper and lower bound on the LLC . . . 3

2 Methodology **4**
2.1 Toy models . . . 4
2.2 When is an eigenvalue zero? . . . 4
2.3 Calculating zero eigenvectors . . . 4

3 Results **4**
3.1 Modular addition transformer . . . 5
3.2 MNIST MLP . . . 5
3.3 Matrix factorization MLP . . . 6
3.4 Summary of results . . . 7

4 Conclusion **7**

A Toy Model Descriptions **9**
A.1 MNIST mlp . . . 9
A.2 Modular addition transformer . . . 9
A.3 Matrix factorisation mlp . . . 9

Abstract

We investigate approximations of the local learning coefficient (LLC), a measure of the model complexity of neural networks, by using a second-order Taylor series expansion of the loss function. We calculate these approximations for a variety of toy models.

1 Introduction

1.1 Statement of Problem

Interpretability aims to identify the algorithms implemented within neural networks by analyzing their parameters and activations. A challenge in interpretability is that neural networks are degenerate: there are many different choices of parameters that implement the same internal algorithm Wei et al. (2023); Watanabe (2009). Multiple parameter configurations can often implement equivalent internal algorithms.

Singular Learning Theory (SLT) quantifies the degeneracy of the loss landscape around a parameter point using the local learning coefficient (LLC). Mathematically, the LLC measures the basin volume of the loss landscape around a given point. Intuitively, this measures the degrees of freedom we have for perturbing the weight in the network's parameter space without effecting the model's behavior.

The LLC provides information about the complexity of the internal algorithm used by the model, and subsequently, the models ability to generalize on new tasks Bushnaq et al. (2024).

Direct computations of the LLC involve an integral that is intractable in practice. Numeric sampling has been used as an attempt to calculate this integral Furman & Lau (2024) Hoogland et al. (2024), though the absolute scale of the resulting LLC values seem to be small [1].

In this paper, we introduce an LLC estimation derived from a Taylor series expansion of the loss function. For a variety of models, we calculate this Taylor series loss approximation up to second-order.

1.2 Background

1.2.1 Power expansion of the behavioral loss

We consider a power expansion of a loss $L(\theta|X)$ in the parameters θ, over a finite dataset X, at some point in parameter space θ^*. Writing this to second order, we have:

$$L(\theta|X) = L(\theta^*) + \frac{\partial L(\theta)}{\partial \theta}\bigg|_{\theta=\theta^*} (\theta-\theta^*) + \frac{1}{2}(\theta-\theta^*)^T \frac{\partial^2 L(\theta)}{\partial \theta^2}\bigg|_{\theta=\theta^*} (\theta-\theta^*) + O(||\theta-\theta^*||^3)\,. \quad (1)$$

Where

$$H(\theta^*) := \frac{\partial^2 L(\theta)}{\partial \theta^2}\bigg|_{\theta=\theta^*} \quad (2)$$

is the network Hessian.

Our chosen loss at a data point x is some convex function[2] $L(y(x), f^{l_{\text{final}}}(x,\theta))$ that depends on the network outputs $f^{l_{\text{final}}}(x,\theta^*)$ at the final layer l_{final}, as well as the correct output labels $y(x)$.

$$L(\theta|X) = \frac{1}{|X|}\sum_{x\in X} L(y(x), f^{l_{\text{final}}}(x,\theta))\,. \quad (3)$$

Typically, we might want the dataset to be the training dataset X, because that is the distribution over which the network's internals were built. So, presumably, all the coherent mechanistic structure in the network will be doing something on some training data points.

[1] From correspondence with the authors

[2] Such as mean squared error or cross entropy loss.

In order to ensure that the network is at a minimum of loss (such that the 0th and 1st order terms of the Taylor expansion are in fact zero, meaning that the Hessian is the first term which can contribute to non-free parameters in the Taylor expansion), we consider not the regular training loss, but the *behavioural loss* of the network (relative to some 'base' network evaluated on the training data), which rather than comparing outputs to the true labels, instead measures the difference in output of a given network to the output from the base network - see Bushnaq et al. (2024) for a full exposition. The behavioral loss is given as:

$$L_B(\theta|\theta^*, \mathcal{D}) = \frac{1}{n}\sum_{x\in\mathcal{D}} ||f_\theta(x) - f_{\theta^*}(x)||^2$$

To be clear, we do not train with this behavioral loss, but use it to estimate the LLC only. Choosing the base network to be the current network of interest, we ensure that our current model is at the global minimum of this behavioural loss.

Therefore, the labels $y(x)$ we're looking for aren't necessarily the training labels. Instead, we set $y(x) = f^{l_{\text{final}}}(x, \theta^*)$, the behavioural loss Bushnaq et al. (2024). We do this because we might be interested in how large a neighborhood around θ^* implements approximately the same map $f^{l_{\text{final}}}(x, \theta^*)$ over the data.

For this project, the behavioral loss is practically convenient, as it ensures that we are at an optimum with $y(x) = f^{l_{\text{final}}}(x)$ for all data points. This makes the terms in our power series easier and cheaper to calculate. In the next sections, we'll see how that works.

1.2.2 Hessian

We now consider a rewritten form of the Hessian. Substituting in our choice of the behavioral loss function and applying the chain rule gives:

$$\begin{aligned}\frac{\partial^2 L(\theta)}{\partial\theta_i\partial\theta_j}\bigg|_{\theta=\theta^*} = &\sum_{x,k,k'} \frac{\partial^2 L(y(x), f^{l_{\text{final}}}(x))}{\partial f_k^{l_{\text{final}}}(x)\partial f_{k'}^{l_{\text{final}}}(x)} \frac{\partial f_k^{l_{\text{final}}}(x,\theta)}{\partial\theta_i}\frac{\partial f_{k'}^{l_{\text{final}}}(x,\theta)}{\partial\theta_j}\bigg|_{\theta=\theta^*} \\ &+ \sum_{x,k} \frac{\partial L(y(x), f^{l_{\text{final}}}(x))}{\partial f_k^{l_{\text{final}}}(x)} \frac{\partial^2 f_k^{l_{\text{final}}}(x,\theta)}{\partial\theta_i\partial\theta_j}\bigg|_{\theta=\theta^*}.\end{aligned} \tag{4}$$

If θ^* is a perfect optimum over the dataset such that $y(x) = f^{l_{\text{final}}}(x, \theta^*)$ for all x, $\frac{\partial L(y(x), f^{l_{\text{final}}}(x))}{\partial f_k^{l_{\text{final}}}(x)} = 0$ for all x, because $L(y(x), f^{l_{\text{final}}}(x))$ is convex. So at a perfect optimum, the second term vanishes, leaving us with:

$$\frac{\partial^2 L(\theta)}{\partial\theta_i\partial\theta_j}\bigg|_{\theta=\theta^*} = \sum_{x,k,k'} \frac{\partial^2 L(y(x), f^{l_{\text{final}}}(x))}{\partial f_k^{l_{\text{final}}}(x)\partial f_{k'}^{l_{\text{final}}}(x)} \frac{\partial f_k^{l_{\text{final}}}(x,\theta)}{\partial\theta_i}\frac{\partial f_{k'}^{l_{\text{final}}}(x,\theta)}{\partial\theta_j}\bigg|_{\theta=\theta^*} \tag{5}$$

For a square loss term, $L(y(x), f^{l_{\text{final}}}(x)) = \left(f^{l_{\text{final}}}(x) - y(x)\right)^2$, this is

$$\frac{\partial^2 L(\theta)}{\partial\theta_i\partial\theta_j}\bigg|_{\theta=\theta^*} = 2\sum_{x,k,k'} \delta_{k,k'} \frac{\partial f_k^{l_{\text{final}}}(x,\theta)}{\partial\theta_i}\frac{\partial f_{k'}^{l_{\text{final}}}(x,\theta)}{\partial\theta_j}\bigg|_{\theta=\theta^*} = 2\sum_{x,k} \frac{\partial f_k^{l_{\text{final}}}(x,\theta)}{\partial\theta_i}\frac{\partial f_k^{l_{\text{final}}}(x,\theta)}{\partial\theta_j}\bigg|_{\theta=\theta^*}. \tag{6}$$

So, to calculate the Hessian, we need to get the gradients of the network outputs $f_k^{l_{\text{final}}}(x)$ with respect to the parameters θ_i over a (representative sample of) the dataset we care about.

1.3 The Hessian rank as an upper and lower bound on the LLC

In a network with d parameters, the network's local learning coefficient is bounded by $\frac{d_1}{2} \leq \lambda \leq \frac{d}{2}$, where d_1 is the rank of the Hessian Watanabe (2009). As proven in Bushnaq (2024), the rank of the Hessian also gives a tight *upper* bound $\lambda \leq \frac{d_1}{2} + \frac{d-d_1}{4}$, where $d-d_1$ is the dimension of the Hessian kernel, meaning the number of zero eigenvalues it has. Putting these together, the dimension of the Hessian d_1 bounds the learning coefficient as

$$\frac{d_1}{2} \leq \lambda \leq \frac{d_1}{2} + \frac{d-d_1}{4}.$$

We are interested in the width of the interval $[\frac{d_1}{2}, \frac{d_1}{2} + \frac{d-d_1}{4}]$. The smaller the width, the better the approximation of the LLC provided by the Hessian rank. Since this width is determined by the number of zero eigenvalues $d - d_1$ of the Hessian, this is the quantity we investigate in the subsequent sections for some example networks.

2 Methodology

2.1 Toy models

We investigate the Hessian approximation of the LLC by estimating the number of zero eigenvalues for the following toy models:

- Modular addition transformer
- MNIST MLP
- Matrix factorization MLP

For details on model architectures and training tasks, see Appendix A

2.2 When is an eigenvalue zero?

Since we're making approximations of these Hessian eigenvalues, our approximations are rarely equal to exactly zero. Therefore, we need to use a "zero eigenvalue cutoff threshold" that specifies when an eigenvalue should be considered a zero eigenvalue.

Each model has it's own scale, range, and distribution of eigenvalues, so the zero eigenvalue cutoff should be unique for each model.

However, in order to make comparisons across various models, this zero-eigenvalue cutoff threshold can alternatively be specified by determining an *accuracy cutoff*, for which, if all zero-eigenvalue eigenvectors are ablated from the model, the test accuracy will remain above.

To facilitate comparisons across different models, we can alternatively define the zero-eigenvalue cutoff threshold using an *accuracy cutoff*. This approach involves ablating the maximum number of bottom eigenvectors one-by-one, up until the model's accuracy drops to a specified level (say, 99.5% of the original accuracy). [3]

Because there isn't an objectively correct zero-eigenvalue cutoff, nor zero-eigenvalue accuracy cutoff, our LLC approximation method is best viewed as calculating the LLC with respect to a particular zero-eigenvalue cutoff, as opposed to determining a single, correct LLC for the model.

2.3 Calculating zero eigenvectors

In order to efficiently compute the zero eigenvalue directions of the Hessian, we use the Lanczos algorithm. Lanczos algorithm utilizes properties of Krylov subspaces in order to efficiently compute the eigenvectors in order of increasing/decreasing eigenvalue magnitude. The full Hessian is never explicitly computed or stored. The storage requirements are the k Arnoldi basis vectors of size N, so $O(Nk)$. The compute requirements are k matrix-vector multiplications, $O(Nk^2)$ and for orthogonalization $O(Nk)$.

3 Results

We now present the results of computationally determining the number of zero eigenvalues for various toy models with respect to a fixed training accuracy cutoff, viz. 99.5% of the original training accuracy. Model performance was measured by projecting the original

[3] Equivalently, one could *project* the model into a basis comprised of the top eigenvectors, until the model reaches that particular training accuracy. In our case, this was more computationally efficient because it relies on calculating top, rather than bottom eigenvectors.

(trained) model onto the subspace of k largest-eigenvalue eigenvectors, with progressively larger k, until the threshold was reached or we ran out of compute resources to calculate additional top eigenvectors.

3.1 Modular addition transformer

Figure 1 shows the accuracy of the modular addition transformer model when projected onto the subspace spanned by the top k largest-eigenvalue eigenvectors. The training accuracy threshold is reached at $k = 1032$, as shown in Figure 2. Hence, we conclude that this model has 86,197 - 1,032 = 85,165 zero eigenvalues.

It is unclear why the accuracy (both training and test) remains relatively flat for the first more than 900 eigenvectors and then rises sharply at around $k = 1,000$. This may require further investigation. For more details on the performance of the original and projected models, see Table 1.

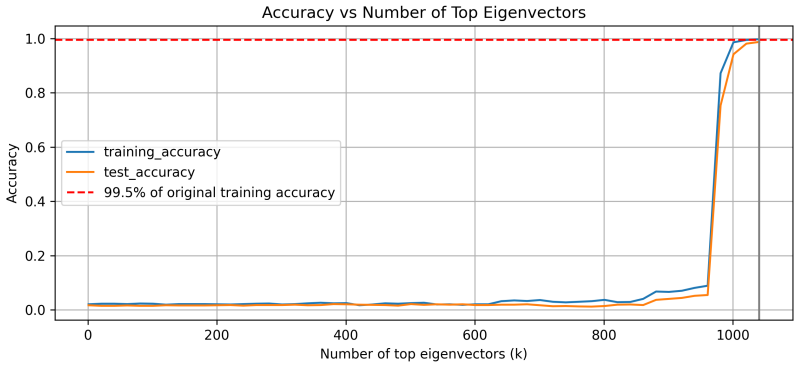

Figure 1: Modular addition transformer: accuracy when projected into top k eigenvectors

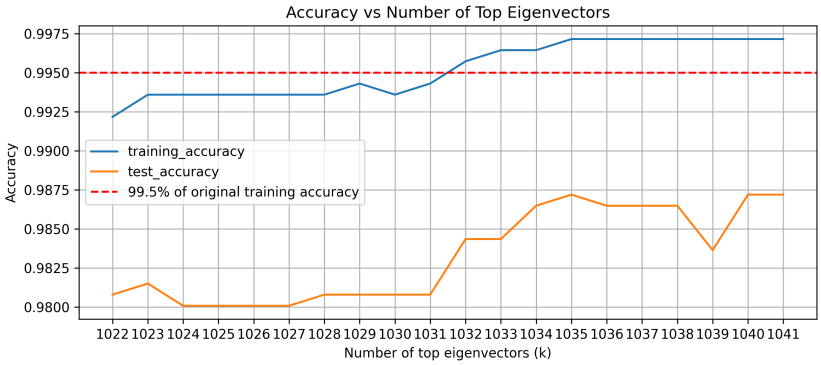

Figure 2: Modular addition transformer: accuracy threshold reached for k = 1032.

3.2 MNIST MLP

Figure 3 shows the accuracy of the MNIST MLP model when projected onto the subspace spanned by the top k largest-eigenvalue eigenvectors. As can be seen, $\frac{dAccuracy}{dk}$ tends to near zero as k reaches 6,000. Beyond this point, the marginal improvement in accuracy (both training and test) by increasing k is very small. Up to $k = 8,000$, the accuracy threshold is not reached. Consequently, we are only able to conclude that the number of non-zero eigenvalues for this model is at least 8,000. Conversely, the number of zero eigenvalues is at most 42,310 - 8,000 = 34,310. For a comparative performance of the original versus projected model see Table 2. (Due to time and resource constraints, we decided to only compute 8,000 eigenvectors.)

Original model dimension	86,197
Number of eigenvectors computed	1,041
Accuracy cutoff reached at k	1,032
Original accuracy on training set	100.00%
Final accuracy on training set	99.72%
Percentage of original training accuracy achieved	99.72%
Original accuracy on test set	100.00%
Final accuracy on test set	98.72%
Percentage of original test accuracy achieved	98.72%

Table 1: Modular addition transformer: original vs. projected model performance

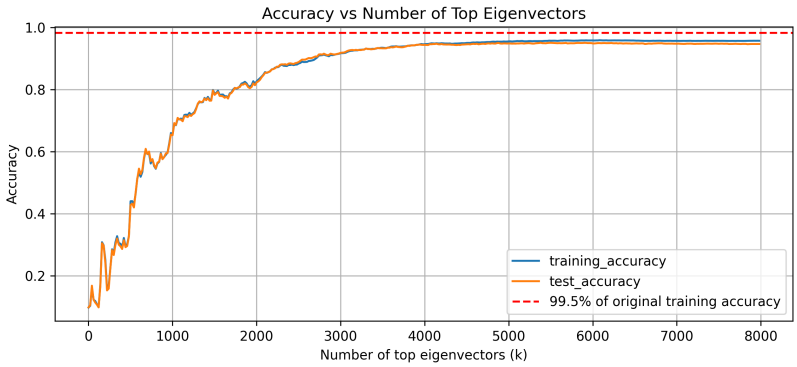

Figure 3: MNIST MLP: accuracy when projected into top k eigenvectors

3.3 Matrix factorization MLP

It is worth noting that zero-eigenvalue cutoffs can be defined in terms of metrics other than accuracy. Here, we use the Mean Squared Error (MSE) to define a zero-eigenvalue cutoff for the matrix factorization task.

Figure 4 shows the mean squared error of the matrix factorization mlp model on the training dataset when projected onto the subspace spanned by the top k largest-eigenvalue eigenvectors of the Hessian. The original model achieved 0.0 MSE on the training dataset. While the projected model does not achieve 0.0 MSE even after including 700 eigenvectors, the MSE curve does become flat at 0.011 after k = 400. This would suggest that the choosing a cutoff of ≈ 0.011 would give a zero-eigenvalue count of 400. This matches the number of zero-eigenvalues calculated analytically, viz. $(l-1) \cdot n^2 = 1 \cdot 20^2 = 400$ (see Appendix A for details).

Original model dimension	42,310
Number of eigenvectors computed	8,000
Accuracy cutoff reached at k	n/a
Original accuracy on training set	98.74%
Final accuracy on training set	95.69%
Percentage of original training accuracy achieved	96.91%
Original accuracy on test set	97.20%
Final accuracy on test set	94.63%
Percentage of original test accuracy achieved	97.35%

Table 2: MNIST MLP: original vs. projected model performance

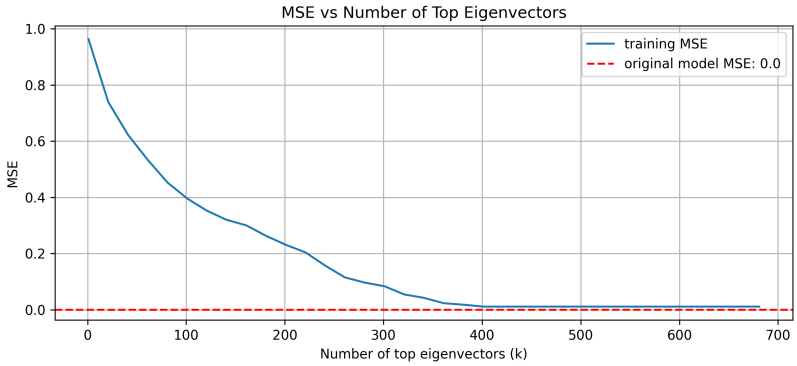

Figure 4: Matrix Factorization MLP: MSE when projected into top k eigenvectors

Original model dimension	800
Number of eigenvectors computed	700
MSE cutoff reached at k	400
Original MSE on training set	0.000
Final MSE on training set	0.011

Table 3: Matrix Factorization MLP: Original vs. projected model performance

3.4 Summary of results

When requiring a 99.5% accuracy threshold for zero-eigenvalue ablation, we get the following bounds on the LLC (Table 4):

$$\frac{d_1}{2} \leq \lambda \leq \frac{d_1}{2} + \frac{d - d_1}{4}$$

Model	**Bound**
Modular Addition Transformer	$516 \leq \lambda \leq 21,807.25$
MNIST MLP	$4,000 \leq \lambda \leq 29,732.5$
Matrix Factorization MLP	$\lambda = 400$

Table 4: Summary of successful LLC bounds

Note that the MNIST MLP bound is based on the computation of a limited number of eigenvectors (viz. 8,000). Computing more eigenvectors will allow us to place a tighter bound on the LLC for this model.

4 Conclusion

Hessian rank provides useful upper and lower bounds for the LLC. This approach offers a computationally feasible method for estimating model complexity across different architectures and tasks. Our findings revealed significant variations in the number of zero eigenvalues across different models.

The accuracy cutoff approach for determining zero eigenvalues allowed for meaningful comparisons between models. Notably, we observed a sharp increase in accuracy at a specific number of eigenvectors for some models, such as the modular addition transformer, suggesting potential critical thresholds in the model's parameter space.

While our study provides valuable insights into neural network complexity, it also has limitations.

Computational constraints prevented the calculation of all eigenvectors for larger models. Additionally, the choice of accuracy cutoff threshold introduces some subjectivity in determining zero eigenvalues.

Our team is currently applying this LLC estimation to a convolutional neural network (CNN) trained on CIFAR 10 image data. The team is also working on extending this power series expansion to 4th order terms, which may provide tighter bounds on the LLC estimation, and lend insight into the degree to which Hessian zero-eigenvectors are interacting at higher orders.

References

Lucius Bushnaq. Hessian rank bounds learning coefficient. https://www.lesswrong.com/posts/Z2RWoyJp9j2yEfbHN/the-hessian-rank-bounds-the-learning-coefficient#T6q29feF5Fjatmdnq, 2024.

Lucius Bushnaq, Jake Mendel, Stefan Heimersheim, Dan Braun, Nicholas Goldowsky-Dill, Kaarel Hänni, Cindy Wu, and Marius Hobbhahn. Using degeneracy in the loss landscape for mechanistic interpretability, 2024. URL https://arxiv.org/abs/2405.10927.

Zach Furman and Edmund Lau. Estimating the local learning coefficient at scale, 2024. URL https://arxiv.org/abs/2402.03698.

Jesse Hoogland, George Wang, Matthew Farrugia-Roberts, Liam Carroll, Susan Wei, and Daniel Murfet. The developmental landscape of in-context learning, 2024. URL https://arxiv.org/abs/2402.02364.

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

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

Sumio Watanabe. *Algebraic Geometry and Statistical Learning Theory*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2009.

Susan Wei, Daniel Murfet, Mingming Gong, Hui Li, Jesse Gell-Redman, and Thomas Quella. Deep learning is singular, and that's good. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):10473–10486, December 2023. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3167409. URL http://dx.doi.org/10.1109/TNNLS.2022.3167409.

A Toy Model Descriptions

A.1 MNIST mlp

The MNIST dataset LeCun et al. (1998) is a well-studied problem in ML, involving classification of a dataset of hand-drawn numeric digits. We train a dense feed-forward neural network with 2 hidden layers of width 50 and ReLU activation function. The network was trained for 10 epochs, achieving a cross-entropy train loss of 0.016 and test loss of 0.246.

A.2 Modular addition transformer

Given a dataset consisting of pairs of numbers (x_{i1}, x_{i2}) both less than some prime p, labelled by their sum modulo p, $y_i = (x_{i1} + x_{i2}) \bmod p$, a network is trained to predict this sum-mod-p. The networks first embed each of the two numbers as vectors, before concatenating these vectors then applying a single hidden layer with ReLU activations, and finally mapping to p output numbers. The networks trained are in general overparametrised (in that they have more parameters than number of training samples) and achieve a perfect accuracy in both training and validation datasets. This setup, described in Power et al. (2022), is commonly used to study grokking, although this feature is not required for our studies.

The model used in our analysis had $p = 53$, with inputs processed through a 53×128 embedding layer, followed by the concatenation of the embeddings processed through a ReLU activated 256×256 hidden layer, followed by a 256×53 output layer. Initialization: bias zero, weights with Xavier

A.3 Matrix factorisation mlp

Given an unknown full-rank $n \times n$ matrix A, a (known) collection of n linearly independent vectors x_i , and (known) vectors $y_i = Ax_i$, we want to learn the matrix A. This is of course a trivial problem if we can invert A, but can provide a useful example of free directions in parameter space. In particular, we create and train a network with l hidden layers each with n neurons and no bias, thus effectively creating a network whose parameters are l matrices $W_1, ..., W_l$, having ln^2 parameters but where $(l-1)n^2$ of these are expected to be redundant. For small n, we can calculate the Hessian for different values of l and determine how many eigenvalues correspond to 'free' directions. For our analysis, we chose $n = 20$ and $l = 2$.