Shape of Minima of Deep Learning Architectures for Sequence Modelling

Sepideh Mamooler, Artur Szalata, Mattia Mariantoni
EPFL, Switzerland

***Abstract*—Training neural networks is a non-convex optimization problem. In general one can only hope to arrive at a good local minimum in such a scenario. Many parameters influence the loss landscape as well as the local optimum that is eventually reached. In this work we apply recently proposed methods to characterize and evaluate the optima that are reached by two dominant deep neural network architectures used in sequence modelling tasks, namely recurrent neural networks and Transformers.**

I. Introduction

Characterization of loss landscape of neural networks is the object of interest of many in the deep learning community ([1], [2], [3], [4], [5], [6], [7], [8], [9] among others). The objective functions of the optimization problems that have to be tackled while training deep neural networks are non-convex and the theoretical guarantees about the popular algorithms and architectures are very scarce. However, such models have empirically achieved state-of-the-art results on a wide variety of tasks.

It is widely believed that one of the reasons behind certain network designs generalizing better than others is the sharpness of minima. Flat minimizers result in smaller generalization error [7]. One of the methods to evaluate sharpness is to analyze the hessian at the minimum [7]. However, it has been shown that these measures are not invariant to certain symmetries in the network and hence are not fully reliable [10]. Another approach is rather qualitative. It relies on visualization techniques to inspect also the neighborhood of the minimizer [11] [12].

Recurrent neural networks such as long short-term memory (LSTM) [13] and gated recurrent neural networks (GRU) [14] are well established models in sequence modeling that have been recently outperformed on many tasks by attention-based models like Transformer [15]. While much work has been devoted to developing ever more complex and effective models [16], to our knowledge the differences in optimization landscape of the two approaches are yet to be explored.

II. Models

A. Recurrent Neural Networks

Recurrent neural networks have the ability to parse sequences of any length. The input sequence elements are passed one-by-one and the information about the previously passed elements of the sequence are stored in hidden states (in case of LSTM also a cell state) vectors that are also used as the input alongside subsequent input sequence parts. Hence, it is the hidden states that play the crucial role in modelling the whole sequences in RNNs.

In our experiments we have used Pytorch [17] implementation of LSTM and GRU. Our particular implementation is presented in figure 5. Linear layer is added before the output to change the dimensions of the hidden state of the last unit to the one desired on the output.

B. Transformer

Transformer is another model used when working with sequential data. We designed an architecture presented in figure 6, consisting of mainly Transformer encoder layers. The use of only encoder layers from Transformer has been popularized by BERT [18]. Similarly to the scenario with RNN we use a linear layer before the output to change the dimensions of the output. We use another linear layer to change the dimensions of the input to match the chosen shape of the embeddings within the Transformer layers.

The dominant element of a Transformer encoder layer is the attention [19] and each layer can use multiple attention heads. Aside from that they use layer normalization [20], residual connections [21], dropout [22] and a feedforward layer.

Within the Transformer each embedding is processed using the same weights (matrices) which makes sequence elements indistinguishable for the network. For that reason we use the absolute positional encoding [15] before feeding the input to the first encoder layer. Note that unlike in the case of RNN, within transformer layers the distance between the input sequence elements makes little difference, because where in LSTM the information from processing one element has to be passed in hidden states through multiple computational steps, attention can directly use any sequence elements while computing every embedding.

III. Tasks

We consider two tasks for training these models: Sequence Classification, and Sequence Learning. The input in both tasks is made up of sequences $S_0, S_1, ..., S_{L-1}$ and targets $y_0, y_1, ..., y_{L-1}$, where L is the dataset size in all our experiments set to 5000. In our experiments we use only sequences on fixed length equal to 5. In all tasks we use 10% of the dataset for testing and 10% of the remaining training set for validation.

A. Sequence Classification

The goal of this task is to classify a sequence based on the sum of its elements. The input consists of sequences

$S_i = [x_{0_i}, x_{1_i}, ..., x_{N-1_i}]$, where N_i is the length of sequence S_i. The target $y_i \in \{0, 1\}$ is defined based on the sum of elements of sequence S_i compared to a threshold T as shown in equation 1. In our experiments we set $T = 0$ and the elements of the sequence are drawn uniformly at random from range (-1000, 1000).

$$y_i = \begin{cases} 0, & \text{if } \sum_{n=0}^{N-1_i} x_{n_i} < T \\ 1, & \text{otherwise} \end{cases} \quad (1)$$

B. Sequence Learning

This tasks aims to predict the first next element of a finite arithmetic sequence. The input consists of sequences $S_i = [x_{0_i}, x_{1_i}, ..., x_{N-1_i}]$, where $x_{n_i} = x_{n-1_i} + C_i$. N_i is the length of sequence S_i and C_i is a constant indicating the increment factor of the arithmetic sequence that in our experiments is fixed at 1000 for all sequences. The first element of each sequence is chosen uniformly at random from range (-1000, 1000). The target is hence $y_i = x_{N-1_i} + C_i$.

IV. Methods

A. Loss Landscape Visualization

In order to visualize the optimization landscape, we use the methods proposed by Goodfellow et al. 2015 [6] and Li et al. 2018 [12].

1) 1D Linear Interpolation: This visualization takes two sets of parameters, the initial parameters of the model $\Theta_{initial}$ and Θ_{final} and computes the loss $J(\Theta)$ at points between the two of those. The points in the parameter space are chosen through linear interpolation between $\Theta_{initial}$ and Θ_{final}. While this is not the path we took while optimizing the model, it is a simple method that can give us intuitive understanding.

2) Contour Plots: In this approach we focus on exploring the neighborhood of Θ_{final}. In particular, we choose two direction vectors δ and η and plot the 2D function $f(\alpha, \beta) = J(\Theta_{final} + \alpha\delta + \beta\eta)$. The choice of the direction vectors δ and η is important due to the fact that network weights are scale invariant when changing the scale of subsequent layers at the same time. This is even more pronounced when using batch or layer normalization that rescale the weights at each layer. To solve that problem we use layer-normalized direction vectors proposed by Li et al. 2018 [12].

B. Sharpness

We have considered three metrics based on the calculation of the hessian of the loss function developed with the PyHessian framework [23]. Note that computing hessian for deep networks is computationally infeasible, hence we can only rely on an approximation. First of the metrics is the eigenvalue with maximum absolute value. The eigenvalues of the Hessian describe the local curvature of the loss landscape and large eigenvalues are associated with slower optimization process. They are also conjectured to negatively influence the generalization capabilities [24]. Additionally, we considered the trace of the hessian, which is equivalent to the sum of the eigenvalues, as a metric. We also took into consideration the plot of the full empirical spectral density (ESD) of the hessian eigenvalues defined as $\phi(t) = \frac{1}{m}\sum_{i=1}^{m} \delta(t - \lambda_i)$ where $\delta(\cdot)$ is the Dirac distribution, λ_i is the i^{th} eigenvalue of H, in descending order, and m is the number of parameters of the model [23]. This metric can show if the spectrum is almost entirely flat which can be an indication of the fact that the loss surface has a low intrinsic dimensionality [25].

V. Experiments

A. Hyperparameters Held Constant

We keep some of the training parameters constant for all the experiments due to the time and computational resources constraints. Another reason is brevity, hence we devote our analysis to the results we observed to be most interesting. The fixed hyperparameters include the use of AdamW optimizer with learning rate 0.001, clipping the gradients to the maximal value 0.25, using batch size of 10. We use mean squared error for sequence learning and binary cross entropy for sequence classification as the loss function.

Apart from that, certain hyperparameters are also held constant across all models. Dropout in RNNs (as indicated in figure 5)) and Transformer encoder layers is kept at 0.1.

B. Sequence Classification

We train all the models (Transformer, LSTM, GRU) for 10 epochs, since we have found that they all appear to converge at that point. Each model is trained for the same set of configurations of layers and hidden embedding dimension. The Transformer model uses varying number of attention heads.

C. Sequence Learning

Here we train only chosen configurations of two models, LSTM and Transformer, for 50 epochs each. Both models use two layers and contain a similar number of tunable parameters (2629 for the transformer and 2647 for the LSTM model). Hidden embedding dimension for Transformer is 12 and for LSTM is 14. Transformer uses a single attention head. We chose those hyperparameters because the resulting models exhibit interesting properties.

VI. Results and Conclusions

A. Sequence Classification

The method for those experiments is described in section V-B. Table I presents the greatest in absolute value eigenvalue of the Hessian (top λ), the Hessian trace and the loss on the test set achieved by the models. Figures 1, 2 and 7 show a comparison for models with different configurations. We observe that the minima of RNNs appear much more sharp (using the top λ and Hessian trace metrics), but at the same time they achieve lower loss on the test set and do so more consistently than Transformers. Qualitative evaluation of the loss landscape around the minima, for the best models in terms of loss shown in figures 3 and 4, shows that quantitative metrics may not capture all relevant information about flatness. Here we observe that the minimum obtained by LSTM is very sharp in certain directions, but it is also near a plateau, while

for the Transformer we observe no flat regions whatsoever. This is consistent with the conviction that flatter minima are generalizing better and demonstrates that there are problems on which it may be preferable to use RNNs over Transformers.

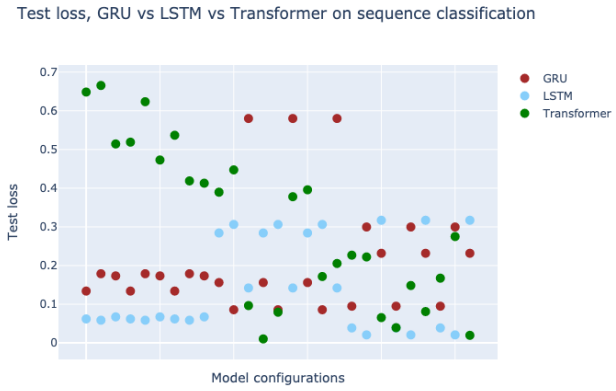


Fig. 1: The order on X axis does not correspond to any particular hyperparameter of the architectures. Note that where for Transformer we varied the number of attention heads the RNN model results are just duplicated. The hyperparameters used and numeric values of the results are presented in table I.

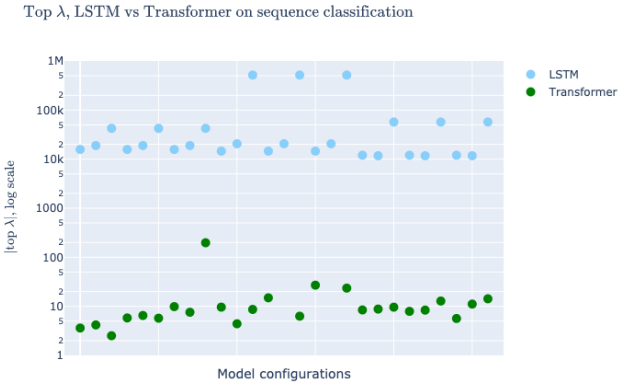


Fig. 2: X axis as in figure 1.

B. Sequence Learning

While RNNs fared better on the simple sequence classification task, we have not observed their convergence on sequence learning task, despite running training for significantly more epochs and varying learning hyperparameters. The loss visualization between initialization parameters and the final parameters for models with the same number of layers and similar number of trainable parameters is presented in figure 8. We see that the loss landscape of both models is rather smooth between the two points, but only Transformer converges to a low loss. Optimization for RNN fails to converge to a comparably low loss. ESD spectrum of Hessian (9) for Transformer indicates that it is a sharp minimum but we observe that the model appears to have converged to a good minimum and generalizes well.

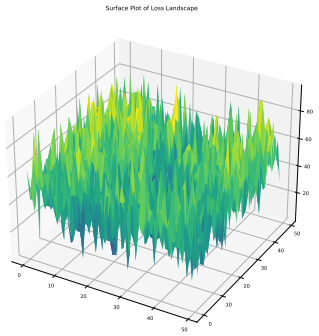


Fig. 3: Transformer loss landscape on sequence classification around the minimum. 3D surface of the loss landscape around the minimum.

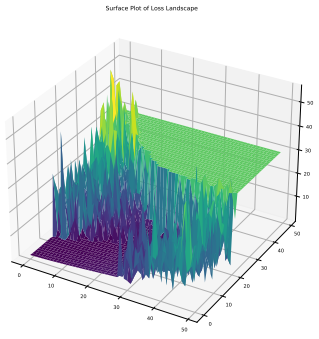


Fig. 4: LSTM loss landscape on sequence classification around the minimum. 3D surface of the loss landscape around the minimum.

C. Conclusion

Our exploration of models used for sequence modeling demonstrates that metrics commonly used to describe the minima of deep networks are not necessarily reliable indicators of the generalization capabilities. This aligns with some scepticism expressed in literature that stress the lack of invariance of such metrics to certain symmetries in the networks [10]. We demonstrated that in some cases qualitative methods for visualizing the loss landscape may correlate better with common beliefs about the superiority of flat optima. Moreover, we have shown that RNN and Transformer models have vastly different optimization landscapes and it is crucial to choose a model according to the specific task.

REFERENCES

[1] H. He, G. Huang, and Y. Yuan, "Asymmetric valleys: Beyond sharp and flat local minima," vol. 32. [Online]. Available: https://papers.nips.cc/paper/2019/hash/01d8bae291b1e4724443375634ccfa0e-Abstract.html

[2] A. Choromanska, Y. LeCun, and G. B. Arous, "Open problem: The landscape of the loss surfaces of multilayer networks," in *Conference on Learning Theory*. PMLR, pp. 1756–1760, ISSN: 1938-7228. [Online]. Available: http://proceedings.mlr.press/v40/Choromanska15.html

[3] Y. Cooper, "The loss landscape of overparameterized neural networks." [Online]. Available: http://arxiv.org/abs/1804.10200

[4] F. Draxler, K. Veschgini, M. Salmhofer, and F. Hamprecht, "Essentially no barriers in neural network energy landscape," in *International Conference on Machine Learning*. PMLR, pp. 1309–1318, ISSN: 2640-3498. [Online]. Available: http://proceedings.mlr.press/v80/draxler18a.html

[5] R. Ge, J. D. Lee, and T. Ma, "Learning one-hidden-layer neural networks with landscape design." [Online]. Available: https://openreview.net/forum?id=BkwHObbRZ

[6] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems." [Online]. Available: http://arxiv.org/abs/1412.6544

[7] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima." [Online]. Available: https://openreview.net/forum?id=H1oyRlYgg

[8] D. Mehta, T. Chen, T. Tang, and J. D. Hauenstein, "The loss surface of deep linear networks viewed through the algebraic geometry lens." [Online]. Available: http://arxiv.org/abs/1810.07716

[9] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou, "Empirical analysis of the hessian of over-parametrized neural networks." [Online]. Available: http://arxiv.org/abs/1706.04454

[10] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *International Conference on Machine Learning*. PMLR, pp. 1019–1028, ISSN: 2640-3498. [Online]. Available: http://proceedings.mlr.press/v70/dinh17b.html

[11] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems." [Online]. Available: http://arxiv.org/abs/1412.6544

[12] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets." [Online]. Available: http://arxiv.org/abs/1712.09913

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," vol. 9, no. 8, pp. 1735–1780. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling." [Online]. Available: http://arxiv.org/abs/1412.3555

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need." [Online]. Available: http://arxiv.org/abs/1706.03762

[16] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners." [Online]. Available: http://arxiv.org/abs/2005.14165

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library." [Online]. Available: http://arxiv.org/abs/1912.01703

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding." [Online]. Available: http://arxiv.org/abs/1810.04805

[19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate." [Online]. Available: http://arxiv.org/abs/1409.0473

[20] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization." [Online]. Available: https://arxiv.org/abs/1607.06450v1

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," p. 30.

[23] Z. Yao, A. Gholami, K. Keutzer, and M. Mahoney, "PyHessian: Neural networks through the lens of the hessian." [Online]. Available: http://arxiv.org/abs/1912.07145

[24] B. Ghorbani, S. Krishnan, and Y. Xiao, "An investigation into neural net optimization via hessian eigenvalue density." [Online]. Available: http://arxiv.org/abs/1901.10159

[25] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, "Measuring the intrinsic dimension of objective landscapes." [Online]. Available: http://arxiv.org/abs/1804.08838

APPENDIX

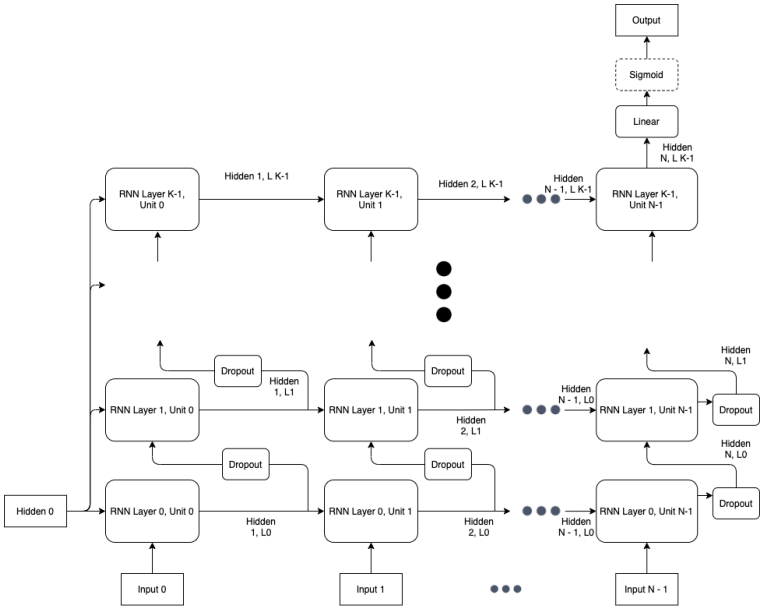


Fig. 5: Our RNN architecture with input of length N and K layers. Sigmoid is only used in sequence classification task from section III-A. Note that all the RNN units in any given layer share weights.

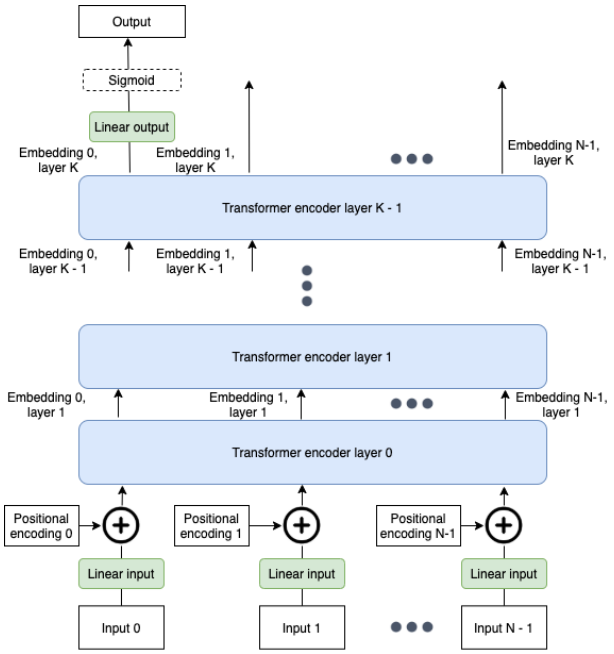


Fig. 6: Our Transformer architecture with input of length N and K layers. Sigmoid is only used in sequence classification task from section III-A. Transformer layers colored in blue have separate set of weights each while the linear layers at the input share weights.

Table I: Sequence classification results. We haven't noted the results for the hessians of GRU, but by looking at the loss landscape plots we are convinced that they are similar to LSTM.

Model	**# layers**	**# heads**	**hidden dim**	**# parameters**	**top λ**	**trace**	**loss**
GRU	1	NA	8	273			0.17
GRU	1	NA	16	929			0.13
GRU	1	NA	32	3393			0.18
GRU	6	NA	8	2433			0.23
GRU	6	NA	16	9089			0.09
GRU	6	NA	32	3507			0.30
GRU	11	NA	8	4593			0.58
GRU	11	NA	16	17249			0.16
GRU	11	NA	32	66753			0.08
LSTM	1	NA	8	361	42418	48958	0.07
LSTM	1	NA	16	1233	18935	22745	0.06
LSTM	1	NA	32	4513	15750	32046	0.06
LSTM	6	NA	8	3241	516844	581415	0.32
LSTM	6	NA	16	12113	14568	14980	0.04
LSTM	6	NA	32	46753	20597	53013	0.02
LSTM	11	NA	8	6121	57102	94000	0.14
LSTM	11	NA	16	22993	12013	21117	0.28
LSTM	11	NA	32	88993	11732	14748	0.31
Transformer	1	1	8	641	2	5	0.51
Transformer	1	1	16	2305	4	6	0.65
Transformer	1	1	32	8705	4	7	0.66
Transformer	1	2	8	641	6	2	0.47
Transformer	1	2	16	2305	6	9	0.52
Transformer	1	2	32	8705	6	9	0.62
Transformer	1	4	8	641	197	171	0.41
Transformer	1	4	16	2305	10	14	0.54
Transformer	1	4	32	8705	7	14	0.42
Transformer	6	1	8	3641	8	15	0.10
Transformer	6	1	16	13425	10	18	0.39
Transformer	6	1	32	51425	4	9	0.45
Transformer	6	2	8	3641	6	-7	0.38
Transformer	6	2	16	13425	15	27	0.01
Transformer	6	2	32	51425	-1792	-2003	0.08
Transformer	6	4	8	3641	23	46	0.20
Transformer	6	4	16	13425	27	51	0.39
Transformer	6	4	32	51425	-11128	-13856	0.17
Transformer	11	1	8	6641	10	-6	0.06
Transformer	11	1	16	24545	8	16	0.23
Transformer	11	1	32	94145	9	15	0.22
Transformer	11	2	8	6641	13	19	0.08
Transformer	11	2	16	24545	8	17	0.04
Transformer	11	2	32	94145	8	21	0.15
Transformer	11	4	8	6641	14	28	0.02
Transformer	11	4	16	24545	6	25	0.17
Transformer	11	4	32	94145	11	17	0.27

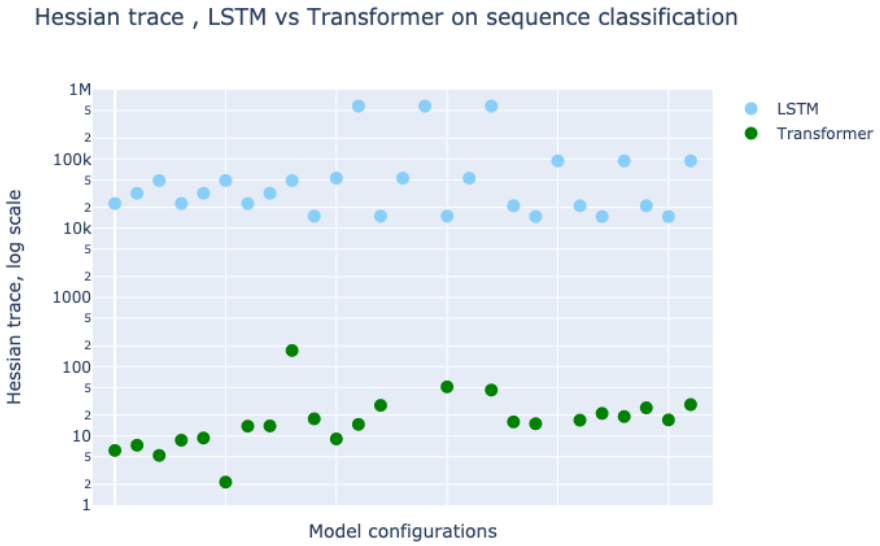
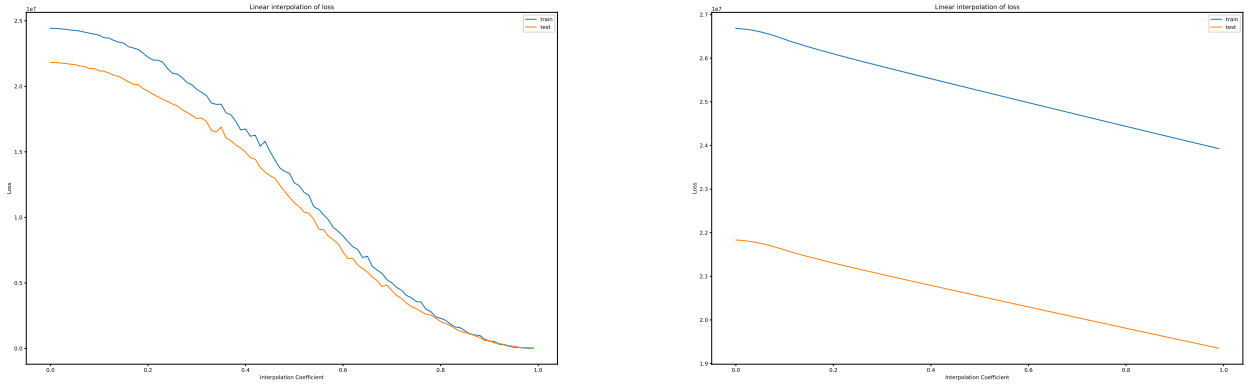


Fig. 7: X axis as in figure 1.



(a) Transformer. Top λ 16709666.

(b) LSTM. Top λ 30.

Fig. 8: Loss interpolation between initialization parameters and after optimization. The model parameters are described in section V-C and the analysis in section VI-B.

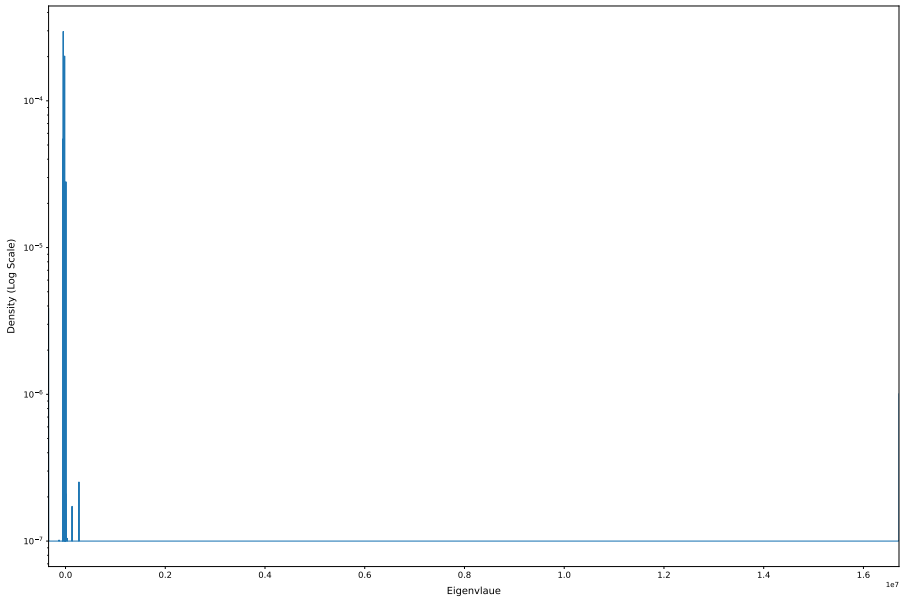


Fig. 9: ESD spectrum of a Hessian for a Transformer. The model parameters are described in section V-C and the analysis in section VI-B.