A Study of the Effects of Model Architecture Ablations on Training Dynamics

Adamo Young
adamo.young@mail.utoronto.ca

Daniel Snider
danielsnider12@gmail.com

Jinyue Feng
jinyue.feng@mail.utoronto.ca

Abstract

In this study we investigate how neural network architecture ablations affect training dynamics. We implement an array of eigenspectrum-based evaluation metrics and look into the effects of activation functions, model depth and width, as well as batch norms on optimization. We conduct our analysis on multi-layer perceptron (MLP) models trained on the MNIST dataset, as well as graph neural networks (GNN) trained on the MolPCBA dataset. Our eigenspectrum metrics seems to capture certain model behaviours, but are not entirely consistent with previous work. We further illustrate advantages and limitations of these metrics on a toy low rank matrix approximation problem. Our work contributes to a public research project called the "Algorithmic Efficiency Benchmark".

1 Introduction

Neural networks have gained tremendous success in a wide range of fields, yet some of the phenomena in neural network training are not fully understood. The typical approach to optimizing hyperparameters involves benchmarking multiple models and selecting the empirically best performing ones. In this project we aim to investigate how modifications to neural network architectures affect training dynamics and apply curvature analysis to examine the loss landscape. The three major aspects we study are activation functions, model sizes, and batch normalization. Our work contributes to a public research project called the "Algorithmic Efficiency Benchmark"[1], of which the primary goal is to rigorously compare machine learning algorithms on their ability to reach convergence faster.

In class we have covered a number of important derivative matrices, including the Hessian, Gauss-Newton Hessian, Classical Gauss-Newton, and Tangent Kernel. One straightforward way to investigate training dynamics is to look at eigenspectra of such matrices. We adopt the stochastic Lanczos quadrature algorithm [Ghorbani et al., 2019] as a promising approach for approximating eigenspectra, due to its computational efficiency. In addition to curvature analysis on MLP models, we also discuss the limitations of the metrics using a toy matrix example.

We start our analysis on MLP models trained on the MNIST dataset and further test our findings on a more complex problem, namely GNN models trained on the MolPCBA dataset. Due to computational constraints, we are only able to compute curvature metrics on the MLP models. An overview of our experiments is shown in figure 1.

The main contributions of our work include:

- A public implementation of various curvature metrics in JAX[Bradbury et al., 2018], building on existing stochastic Lanczos tools

[1] https://mlcommons.org/en/groups/research-algorithms/

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

- A discussion on the limitations of these metrics using a simple low-rank matrix approximation case study
- Extensive architecture ablation experiments using a setup similar to the Algorithmic Efficiency Working Group, with curvature analysis

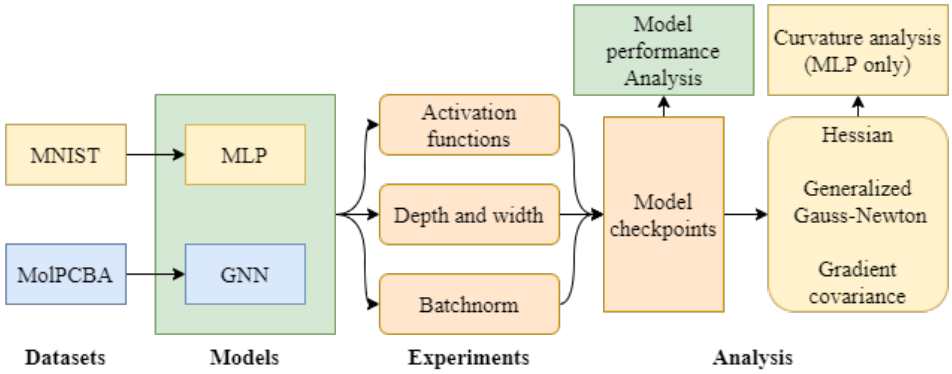


Figure 1: Experiment diagram

2 Curvature metrics

2.1 Matrices of Interest

The neural network training problem can be viewed as a minimization of a loss function $\mathcal{L}(w)$ with respect to the model parameters $w \in \mathbb{R}^M$. Typically, the loss is an empirical average over some dataset $\{x_i\}_i^N$,

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(f(\theta, x_i), x_i) \tag{1}$$

where f is a function parameterized by w.Typically, w is very high dimensional ($M >> 10000$).

There are a number of matrices that can provide useful information about optimization dynamics. The Hessian matrix $\nabla_w^2 L(w)$ is most commonly studied, as (in theory) it provides perfect second-order information about the optimization problem. Since w is high-dimensional, it is generally inadvisable to compute the full Hessian, but calculating Hessian-vector products is efficient using automatic differentiation. This means we can approximate the eigenspectrum using stochastic Lanczos quadrature Ghorbani et al. [2019], which only requires a series of matrix-vector products.

The Generalized Gauss-Newton (GGN) is a PSD matrix that in some sense approximates the Hessian. Let $z = f(w, x)$ be the output of the model f on the input x, and let $J_{zw} = \nabla_w f(w, x)$ and $H_z = \nabla_z L(z, x)$. Then the GGN is defined as $J_{zw}^T H_z J_{zw}$.

The final matrix of interest is the Gradient Covariance matrix (Σ, Ghorbani et al. [2019]), which is simply the following:

$$\Sigma(w) = \frac{1}{N}\sum_{i=1}^{N} \nabla_w L(f(w, x_i), x_i)[\nabla_w L(f(w, x_i), x_i)]^T \tag{2}$$

Intuitively, it is the covariance matrix formed from the average of the N mini-batch gradient covariances. As a covariance matrix, by definition it is also PSD.

Like the Hessian, GGN-vector products and Covariance-vector products can be efficiently computed using standard automatic differentiation tools. Building on an existing Stochastic Lanczos implementation in JAX (from this repo), we were able to efficiently approximate the eigenspectra of these three matrices.

2.2 Metrics of Interest

We have assembled a set of metrics to analyze the eigenspectrum of a matrix. For each metric, we briefly describe how to compute it and what kind of information it can provide:

- **Max Eigenvalue**: The largest magnitude positive eigenvalue, gives a notion of the peak sharpness of the loss surface, since there are typically more large positive eigenvalues than negative ones. Used frequently in spectral analysis Ghorbani et al. [2019], Gilmer et al. [2022]
- **Min Eigenvalue**: the largest magnitude negative eigenvalue, generally useful at the beginning of training (when there are still sharp directions to descend in), not useful for PSD matrices or later on in training. Used frequently in spectral analysis Ghorbani et al. [2019], Gilmer et al. [2022]
- **Max / Top-K Eigenvalue Ratio (MER-K)**: the ratio of the largest to Kth largest positive eigenvalue, captures how much of an outlier the maximum eigenvalue. Generally, the presence of large outlier directions is undesirable. Used in Ghorbani et al. [2019].
- **Trace / Top-K Eigenvalue Ratio (TER-K)**: The ratio of the trace of the matrix with respect to the sum of the top K largest eigenvalues. Only useful for PSD matrices. The trace is approximated using the trace of the Lanczos tridiagonal matrix. If this ratio is approximately 1, it means that the matrix is close to rank K. Having a rank 1 GGN matrix is sometimes associated with the vanishing gradient problem.
- **Positive / Negative L1 Energy Ratio (PNR)**: The ratio of positive L1 energy to negative L1 energy, where positive/negative L1 energy is the integral of the (normalized) positive/negative eigenspectrum, respectively. Having higher Negative L1 Energy (but small Min Eigenvalue) is associated with being closer to the end of optimization. Used in Ghorbani et al. [2019].
- **Projected / Full Gradient Energy Ratio (PGR-K)**: The ratio $\frac{\|P_K \nabla_w \mathcal{L}(w)\|_2}{\|P_L \nabla_w \mathcal{L}(w)\|_2}$, where P_K is a projection onto the subspace spanned by the top K eigenvectors of the L-dimensional tridiagonal Lanczos matrix and P_L is a projection onto the subspace spanned by all L eigenvectors of the tridiagonal matrix. If the gradient energy is concentrated in a small space, this is undesirable, since the smaller directions have been empirically shown to be important for achieving good solutions. Similar to a metric from Ghorbani et al. [2019].

3 Case Study: Low Rank Approximation

To assess the utility of some of the aforementioned curvature metrics, we decided to study them in a well-characterized toy problem. Inspired by comments from Justin Gilmer (scientist at Google Brain and member of the Working Group), we looked at the problem of fitting a low-rank matrix approximation using (full batch) gradient descent. Our formulation of the problem involves learning two matrices $L, R \in \mathbb{R}^{N \times D}$, where $D << N$ (in our case, $D = 10$ and $N = 100$), whose matrix product approximates the matrix $M \in \mathcal{S}^N$:

$$\min_{L,R} \|LR^T - M\|_2 \tag{3}$$

Of course, this problem has a well-known closed-form solution, namely the rank D singular value decomposition of M [Eckart and Young, 1936].

In our experiment, we sample our target M from the distribution of symmetric matrices with independent unit Gaussian entries. When L, R are initialized reasonably (i.e. with values drawn IID from a unit Gaussian), the problem can be solved quite quickly with simple gradient descent. However, changing the initialization of L and R, for instance by sampling the values of L and R from distributions with different means, can affect the conditioning of the problem and have a negative impact on the effectiveness of gradient descent. Good performance can be recovered by dramatically lowering the learning rate or by using a "warmup" learning rate scheduler that increases the learning rate in the initial stages of training before slowly decaying it to a low value.

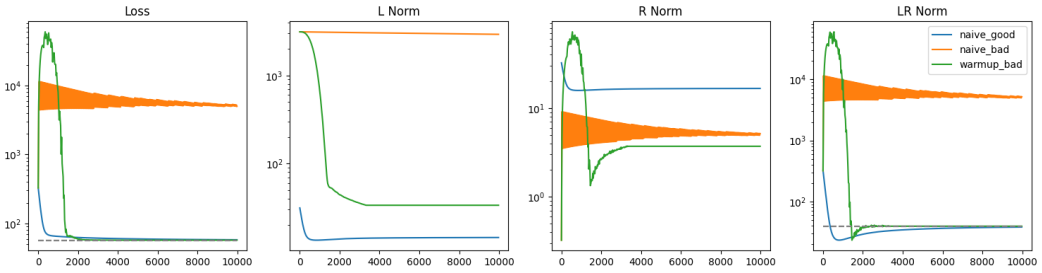


Figure 2: Loss and Frobenius norms (for L, R, and LR^T) at every optimization step. Blue is naive gradient descent with good initialization, Orange is naive gradient descent with bad initialization, and Green is warmup gradient descent with bad initialization. Dashed lines are optimal values computed with SVD. Note that the Orange line is oscillating with a period of about 10 steps.

Figure 2 demonstrates the optimization dynamics of naive gradient descent with a good initialization, naive gradient descent with a bad initialization, and learning rate warmup with a bad initialization. The warmup method performs best overall, achieving near optimal loss when compared to the singular value decomposition. The naive method performs well with a good initialization, but struggles in the more ill-conditioned problem (it begins to oscillate). This plot also gives some intuition as to why warmup performs better: the larger learning rate allows for the L matrix to descend rapidly in a direction that reduces its norm. While this temporarily results in a large loss, it allows the model to escape the poor initialization and find a better local optimum that is closer to the global one. This behaviour is reflected by the LR Norm plot, which measures $\|LR^T\|_2$ at each optimization step.

How well do our curvature metrics capture the poor conditioning of the problem? In Figure 3, we consider two of our standard metrics: the maximum eigenvalue of the Hessian, the ratio maximum to 10th largest eigenvalue of the Hessian. The maximum eigenvalue gives us information about the maximum sharpness of the Hessian. As expected, at initialization the poorly conditioned problems have a much sharper loss landscape. Furthermore, the warmup method rapidly transitions to a smoother loss landscape, while the naive method remains in a relatively sharp loss region.

The Hessian eigenvalue ratio does not appear to provide much information, staying roughly around 1 throughout optimization for all three experiments. One might expect there to be outlier eigenvalues in the Hessian spectrum, but this is not the case. However, in this case the Hessian eigenspectrum is actually bimodal. The higher end of the spectrum is dominated by eigenvalues that are of similar magnitude to the leading singular values of L (10^3). This illustrates how it may be possible for the Hessian to have many sharp directions if a large block of parameters (i.e. the parameters of the matrix L, or the parameters of a single layer in a neural network) is stuck in a sharp region. Of course, increasing K could solve this problem, with the burden of additional Lanczos iterations. Additionally, if we consider the partial Hessian with respect to L and R, we can see in the ill-conditioned case that the maximum eigenvalues differ greatly, which correctly identifies this issue. Depending on the metric, it may be useful to consider analyzing eigenspectra of the Hessian with respect to specific subsets of parameters.

4 Experiments

The following experiments are implemented in JAX[Bradbury et al., 2018]. For each dataset, we set up a performance threshold as the early stopping trigger and treat the total number of trained steps as a measure of training efficiency. We use Weights & Biases[Biewald, 2020][2] to visualize and analyze model performances. For each model, the Lanczos algorithm was run on every checkpoint, computing the eigenspectra for the Hessian, GGN, and Covariance matrices, and all of the metrics from Section 2.2 were computed. All metrics are averaged over three independently trained models. Our complete results are available for your viewing pleasure here.

[2]https://wandb.ai/site

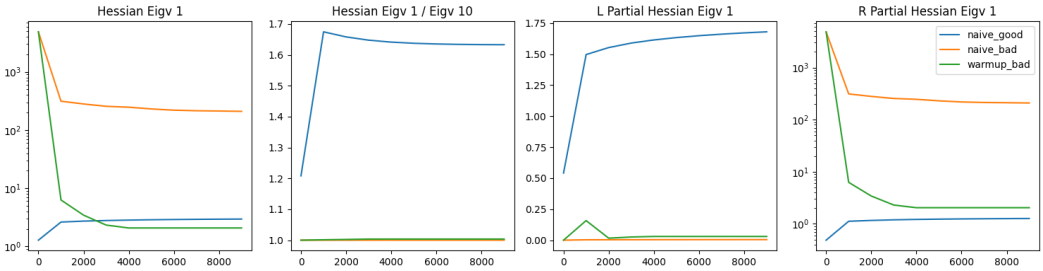


Figure 3: Different Hessian curvature metrics, recorded at intervals of 1000 steps. Blue is naive gradient descent with good initialization, Orange is naive gradient descent with bad initialization, and Green is warmup gradient descent with bad initialization. Note that for this toy example, the eigenspectra were computed exactly using an eigendecomposition.

4.1 MLP

To understand the relationships between architecture changes and training dynamics, we start with in-depth experiments on the MNIST dataset. Our baseline model is a 3-layer MLP with a hidden dimension of 100 units and no batch normalization. We use Adam optimizer and a 1024 batch size. The series of modifications are listed in table 1. We train each model until they reach an accuracy of 0.87.

Table 1: Summary of MLP experiments

Name	Value	Model Width	Model Depth
Batchnorm	0 / 1	100	3 / 5 / 7
Activation function	ReLUs / GELUs / sigmoid / hard_tanh	100 / 200	3

4.1.1 Activation function

In this section we compare the performances of four popular activation functions with two settings of model widths. The activation functions are Sigmoid ($f(x) = \frac{1}{1+e^{-x}}$), Hard Tanh ($f(x) = max\{min[x, 1], -1\}$), ReLU ($f(x) = max(0, x)$), and approximate GELU ($f(x) = \frac{x}{2}(1 + tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3))$) [Hendrycks and Gimpel, 2020]. The results are shown in table 2 (for more detailed training curves, see Appendix or our Weights and Biases page).

Table 2: Steps to reach target accuracy (activation/width)

Activation function	Width	Steps
GELU	200	41
ReLU	200	46
Hard Tanh	200	59
ReLU	100	81
GELU	100	82
Sigmoid	200	89
Hard Tanh	100	167
Sigmoid	100	183

Table 3: Steps to reach target accuracy (batch normalization/depth)

Batch normalization	Depths	Steps
True	3	45
True	5	62
True	7	76
False	3	82
False	7	84
False	5	104

In general, it takes the 200-unit models roughly half the time to reach our target accuracy compared to their 100-unit counterparts, indicating a positive correlation between model width and training efficiency. GELU and ReLU reached top performances while Sigmoid trains significantly slower than all three other functions. Even with a smaller model width, ReLU and GELU still outperforms Sigmoid, which is the most susceptible to the vanishing gradient problem. We also observe a tendency

for GELU to perform better than ReLU in larger models, which agrees with the success of GELU in state-of-the-art transformer-based language models.

Note that the magnitudes of the losses are strongly affected by the natures of the activation functions. More specifically, Sigmoid and Hard Tanh are centred around 0 while ReLU and GELU are not. Without batch normalization, it is expected to see higher losses with ReLU and GELU even when the model performances are better. We examine the effect of batch normalization in the next section.

We wanted to answer two specific questions about the activation functions, using our curvature metrics:

1. Do sigmoid (and to a lesser degree, htanh) models suffer from vanishing gradients?
2. Do sigmoid/htanh models (which are both symmetric) behave more similarly to each other than ReLU/GeLU models (which are asymmetric)?

The answer to the first question appears to be no. Figure 4 displays a selection of curvature metrics that address these questions. The GGN TER-1 metric indicates that none of the models had a GGN of approximately rank 1, which is associated with vanishing gradient. In fact, the activation function most famous for vanishing gradient, sigmoid, had a TER-1 that grew dramatically throughout training. [3] It seems like our metrics were not useful in detecting vanishing gradient. In many respects, the GeLU and ReLU models behave similarly. Interestingly, they appear to have stronger outliers in the Hessian spectrum, and a higher proportion of gradient energy concentrated in a smaller subspace.

Curiously, throughout all of our experiments we were not able to get meaningful signal from the Covariance PGR, even when considering different numbers of dimensions for the projection space (i.e. varying K from 1 to 10). However, we were able to notice patterns in the Hessian PGR, which is strange since one would expect the subspace defined by these two matrices to be quite similar Ghorbani et al. [2019].

4.1.2 Batch normalization

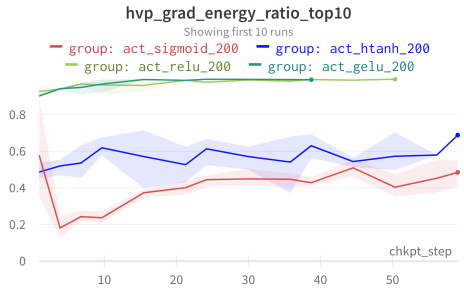
Batch normalization is originally proposed to accelerate neural network training by reducing internal covariate shift and regularization [Ioffe and Szegedy, 2015], however its true mechanism of improving training efficiency is not fully understood. Santurkar et al. [2018] refute the correlation between internal covariate shift and training performances and instead argue that normalization makes the optimization landscape smoother. As shown in table 3(or our Weights and Biases page), we indeed observe that models with batch normalization train faster than those without normalization. In contrast to what we find about model width, increasing model depth does not benefit training efficiency even if the model is supposedly holding more information. We expect that batch normalization to improve the performances of deeper models but one interesting observation is that batch normalization seems to exhibit a greater accelerating effect on shallower models than on deeper models. This may have resulted from noises but still worth noting.

In terms of curvature metrics, we tried to answer the following questions, based on observations from Ghorbani et al. [2019]:

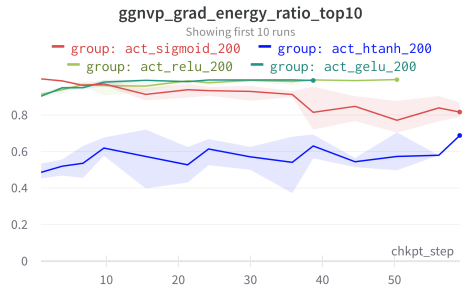
- Does batch norm reduce the maximum sharpness of the loss?
- Does batch norm reduce the number of eigenvalue outliers and spread out the gradient energy?
- Is there a larger density of negative eigenvalues near the end of training (across all models)?

Our findings are summarized in Figure 5. It appears that batch norm does indeed decrease sharpness, in terms of lowering the maximum eigenvalue of the Hessian and GGN. The number of Hessian eigenvalue outliers (as measured by MER-10, not shown) did not behave as expected. Two of the batch norm models (the 5 layer and 7 layer networks) had extremely large outliers once or twice during optimization, but were otherwise very similar to the no batch norm models. Interestingly, the models without batch norm had higher Hessian and GGN PGR-10, which indicates more gradient energy concentrated in a smaller subspace. Finally, the density of negative eigenvalues seemed to

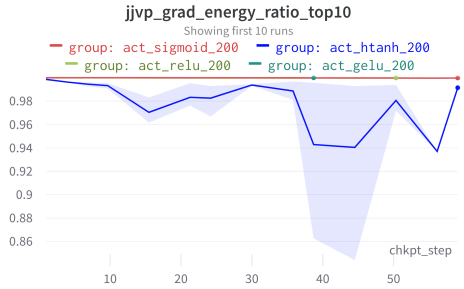
[3] We also verified that the sigmoid model had a lower average gradient norm than the other methods throughout training, which is indicative of vanishing gradient.



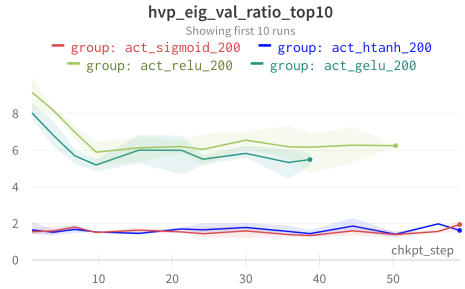
(a) Hessian PGR-10



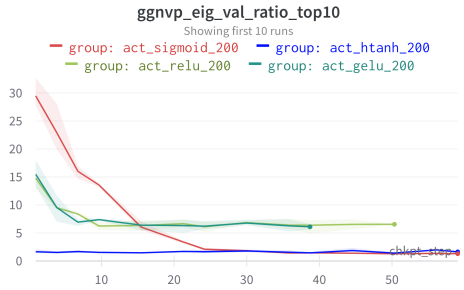
(b) GGN PGR-10



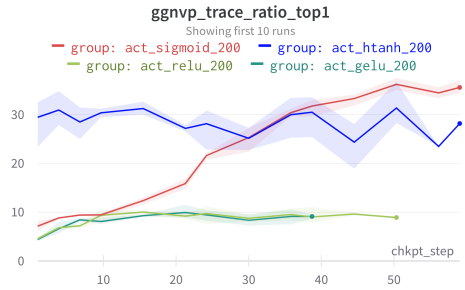
(c) Covariance PGR-10



(d) Hessian MER-10



(e) GGN MER-10



(f) GGN TER-1

Figure 4: Various curvature metrics for the activation experiments, only considering models with width of 200. Red is sigmoid, blue is hard tanh, light green is ReLU, dark green is GeLU. Lines are averages over 3 different runs, with shaded regions indicating range.

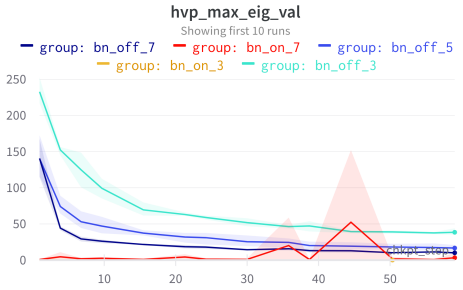
vary more for models without batch norm, but did not seem to vary meaningfully over the course of training in either type of model.

4.2 GNN

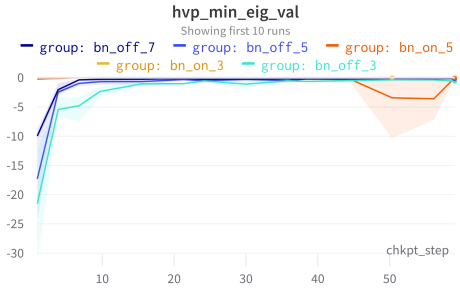
To further our understanding of the relationships between architecture changes and training dynamics, we'll repeat some of our analysis on a GNN model Kipf and Welling [2016]. The GNN is applied to the molecular property prediction task "molpcba" Hu et al. [2020]. This task aims to predict multi-class classification of whether input proteins interact with certain proteins.

Our GNN code [4] is based on the flax reference implementation[5]. The reference GNN implementation uses Adam optimizer with a 2048 batch size and a constant learning rate of 0.001. The reference GNN model architecture consists of 256 latent dimensions in the feature embedder, 256 hidden dimensions, and 5 message passing steps which we consider to be analogous to model depth.

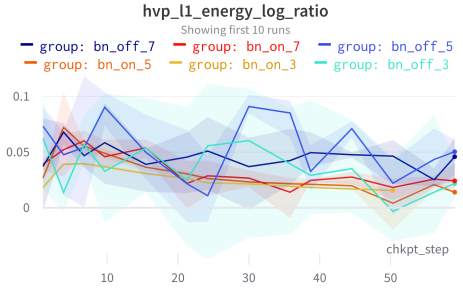
[4] https://github.com/UofT-EcoSystem/algorithmic-efficiency/tree/wandb
[5] https://github.com/google/flax/tree/main/examples/ogbg_molpcba



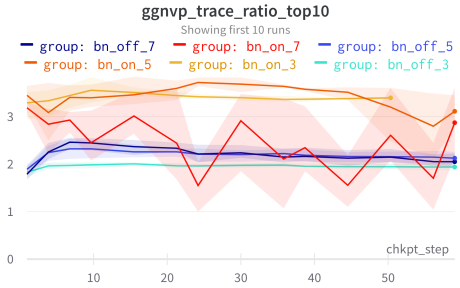
(a) Hessian Max Eigenvalue



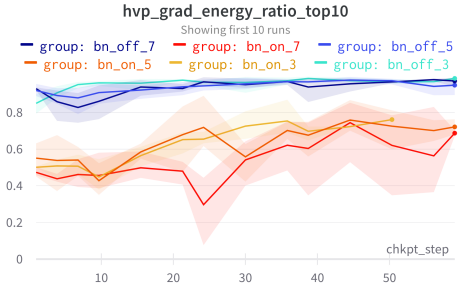
(b) Hessian Min Eigenvalue



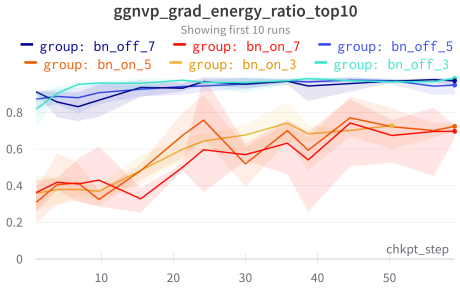
(c) Hessian Log PNR



(d) GGN TER-10



(e) Hessian PGR-10



(f) GGN PGR-10

Figure 5: Various curvature metrics for the batch norm experiments. Warm colours are with batch norm on, cold colours are without batch norm. The 5-layer model with batchnorm on was removed from the subfigure (a) since it spiked once at a very high value; the 7-layer model was removed from subfigure (b) for the same reason.

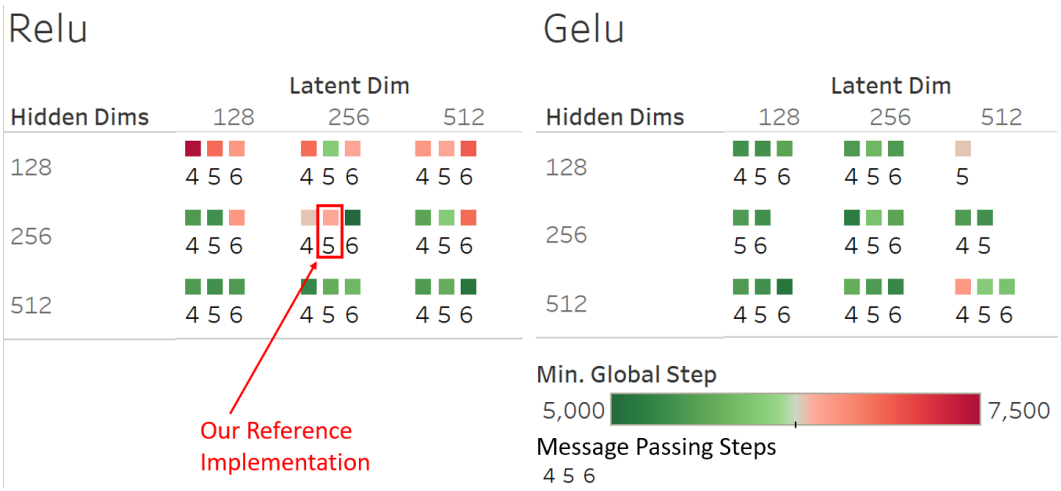


Figure 6: Steps to target (0.24 mAP) for various GNN model architectures. Varying activation, latent dimensions (feature encoder), hidden dimensions, and message passing steps.

It takes 1.5 hours to reach the target 0.24 mean average precision (mAP) on 4x RTX 2080Ti GPUs. We utilized the vector slurm cluster to utilize up to 300 GPUs at a time to complete all our GNN experiments overnight. While we originally intended to perform spectral analysis on the GNN models, it quickly became apparent that this would not be computationally feasible with our current resources and time constraints.

Table 4: Summary of GNN experiments

Name	Value	Encoder Width	Model Width	Depth	LR
Layernorm	0 / 1	256	256	4 / 5 / 6	.001
Activation fn	ReLU / GELU	128 / 256 / 512	128 / 256 / 512	4 / 5 / 6	.0001 / .001 / .01

4.2.1 Activation function

In this section we compare the performance of four popular activation functions with several settings of model size as described in table 4. Overall in figure 6, we found model variations that can learn at better or worse rates than the reference (for more detailed training curves, see figure 10 in the Appendix or our Weights and Biases page). Most of these experiments have multiple repeat runs, however some gaps exist due to crash/preempt/non-start.

We found the activation functions Sigmoid and Hard Tanh did not learn effectively in our setup. We observed their losses reduce from 0.6 to 0.06 after 100 steps then hover around there. Consistent with our results from the MNIST experiments, GELU activation generally outperformed ReLU in terms of training efficiency.

4.2.2 Layer normalization

After removing layer normalization, our models fail to converge (see Figure 9 in the Appendix or our Weights and Biases page for detailed training curves.) Unfortunately, we could not conduct curvature analysis on the GNNs due to computational constraints and explain the seemingly crucial role of layer normalization in our experiments. Our findings also contradict Cai et al. [2021]'s claim that LayerNorm does not have a significant effect on GNN training.

5 Limitations

We are able to derive interesting findings from the MLP models but find it much more difficult to analyze results for the GNN experiments. For example, we observe effects of widths and depths on

MLP training efficiency, but no clear pattern on GNN. When testing with normalization, we find that GNN models without layer normalization fail to learn. A more comprehensive set of experiments involving different normalization methods, for example Graph Norm proposed by Cai et al. [2021], will offer better insights on the effect of normalization in GNN training dynamics. The fact that our findings on MLP did not entirely generalize to GNN also indicate the challenge of understanding neural network optimization. Given the limited number of trials and settings, we need to be careful to draw any concrete conclusions even on the simpler models.

We have highlighted some of the limitations of eigenspectrum metrics in Section 3. Fundamentally, Hessian spectrum analysis makes an assumption that the loss landscape can be modelled with a second order Taylor approximation. Unfortunately, recent work Bai et al. [2020] has shown that this assumption may not be accurate for many real-world models and datasets. Another challenge is the accuracy of the Lanczos approximation. While Stochastic Lanczos quadrature has theoretical guarantees about the fidelity of the eigenspectrum density approximation Ghorbani et al. [2019], these guarantees depend on hyperparameter settings (such as the noise of the Gaussian kernel estimate) that may be difficult and expensive to tune. Most matrices of practical interest have many small magnitude eigenvalues, which are more difficult to estimate accurate. This makes it tricky to measure certain important quantities, such as the condition number of a large PSD matrix, that depend on the minimum magnitude eigenvalue. The entire Lanczos process can be slow and computationally intensive, so even if these small eigenvalues can be accurately estimated, actually computing them is impractical.

6 Related Work

Loss Hessian for understanding neural networks optimization Ghorbani et al. [2019] applied the Lanczos algorithm to visualize the eigenspectra of the Hessian and the classical Gauss-Newton throughout training. They applied spectrum integration metrics that allowed them to compare relative density of positive and negative eigenvalues in the matrix. Using multiple architectures, they observe that the Hessian spectrum is initially dominated by strongly negative eigenvalues. This is also supported by the work of Li et al. [2018], who argued that the loss surface is effectively low-dimensional.

Adams et al. [2018] developed an alternate approach for eigenspectrum estimation that relies on Chebyshev polynomials, and applied their method to compute Laplacian spectra for large networks. However, Ghorbani et al. [2019] empirically demonstrated that the Lanczos approach scales better with larger models.

Gilmer et al. [2022] studied the evolution of maximum eigenvalue of the loss Hessian and demonstrated its critical role in optimization. They found that architecture choices and learning strategies that reduce the loss sharpness early in training benefited optimization.

Neural network architecture choices Currently, there is considerable disagreement in the community regarding the effects of certain architecture choices. For example, Li et al. [2018] hypothesized that adding residual connections has a smoothening effect on the loss landscape. They supported this hypothesis by producing 2D contour plots of loss landscapes for networks with and without residual connection, using the technique of filter-normalization to deal with the issue of ReLU network scale invariance. However, their results disagree with experiments from Ghorbani et al. [2019], who show that the addition of residual connections causes the appearance of high-magnitude positive eigenvalues in the Hessian eigenspectrum, which indicates a sharper loss surface.

Ghorbani et al. [2019] further demonstrated that batch normalization has a positive effect on training by reducing the presence of high-magnitude positive eigenvalues. They also investigated the norm of the Jacobian to show that, without batch normalization, the majority of Jacobian energy is concentrated in directions that are spanned by strongly-positive Gauss-Newton eigenvalues.

As for the effect of model width, Shallue et al. [2019] show in figure 15(d) that for an MNIST MLP more width contributes to reaching a target accuracy in fewer steps. This agrees with our finding in both MLP (any activation) and GNN (relu) models.

7 Conclusion

In this study, we conducted a series of experiments to understand the effects of model architectural changes on neural networks training dynamics. We empirically measure the training efficiency by setting up target performance, and analyze the optimization landscape using a set of eigenspectrum-based metrics. We are aware of the limitations and the size of our study, and only conclude our findings without generalization to other contexts. Among the four activation functions (GELU, ReLU, Hard Tanh and Sigmoid), we find that GELU and ReLU consistently outperform the other two functions. GELU also seems to achieve better training efficiency than ReLU especially with more complex models. For MLPs, increasing model width benefits training efficiency yet adding model depths may hinder the performance. Normalization significantly improves training speed, and may even affect the model's ability to learn.

Through our analysis with curvature metrics, we were able to recover some previously observed behaviour in regards to the addition of batch norm, namely the reduced sharpness of the loss landscape. We were also able to observe how models with similar activations (ReLU and GeLU) behaved similarly. Some of the other metrics, in particular those relating to eigenvalue ratios, L1 energy ratios, and those that depended on the Covariance matrix, were less useful in our analysis. We highlighted some of the limitations of these metrics by applying them to a low rank matrix approximation problem.

References

References

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. *arXiv:1802.03451 [stat]*, Feb 2018. URL `http://arxiv.org/abs/1802.03451`. arXiv: 1802.03451.

Yu Bai, Ben Krause, Huan Wang, Caiming Xiong, and Richard Socher. Taylorized training: Towards better approximation of neural network training at finite width. *arXiv:2002.04010 [cs, stat]*, Feb 2020. URL `http://arxiv.org/abs/2002.04010`. arXiv: 2002.04010.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. *arXiv:2009.03294 [cs, math, stat]*, Jun 2021. URL `http://arxiv.org/abs/2009.03294`. arXiv: 2009.03294.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, Sep 1936. ISSN 1860-0980. doi: 10.1007/BF02288367.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. *arXiv:1901.10159 [cs, stat]*, Jan 2019. URL `http://arxiv.org/abs/1901.10159`. arXiv: 1901.10159.

Justin Gilmer, Behrooz Ghorbani, Ankush Garg, Sneha Kudugunta, Behnam Neyshabur, David Cardoze, George E Dahl, Zack Nado, and Orhan Firat. A loss curvature perspective on training instability in deep learning. page 22, 2022.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415 [cs]*, Jul 2020. URL `http://arxiv.org/abs/1606.08415`. arXiv: 1606.08415.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. 2020. doi: 10.48550/ARXIV.2005.00687. URL `https://arxiv.org/abs/2005.00687`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167 [cs]*, Mar 2015. URL `http://arxiv.org/abs/1502.03167`. arXiv: 1502.03167.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2016. doi: 10.48550/ARXIV.1609.02907. URL `https://arxiv.org/abs/1609.02907`.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv:1804.08838 [cs, stat]*, Apr 2018. URL `http://arxiv.org/abs/1804.08838`. arXiv: 1804.08838.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? May 2018. doi: 10.48550/arXiv.1805.11604. URL `https://arxiv.org/abs/1805.11604v5`.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *arXiv:1811.03600 [cs, stat]*, Jul 2019. URL `http://arxiv.org/abs/1811.03600`. arXiv: 1811.03600.

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. Note that the Reference section does not count towards the page limit.

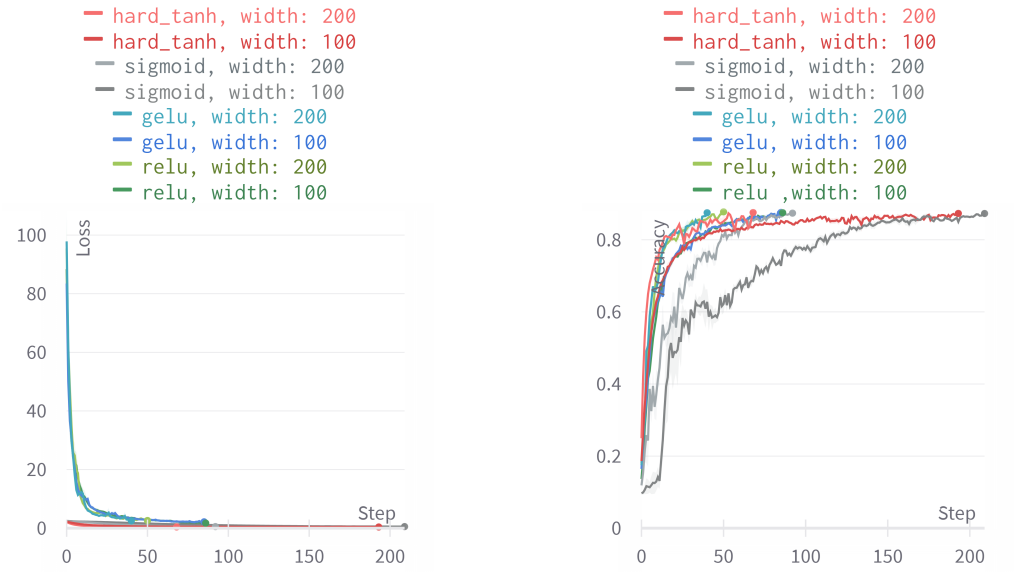
[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

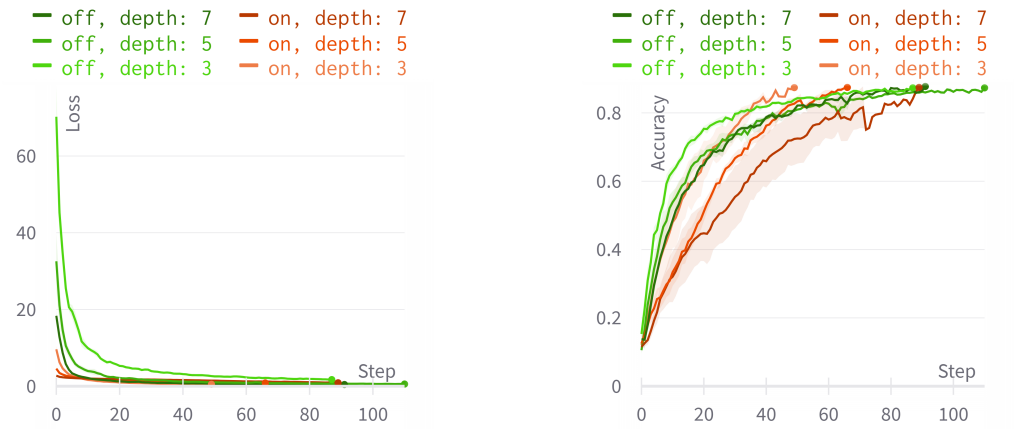
A Appendix

We include training curves of all the experiments in the appendix.



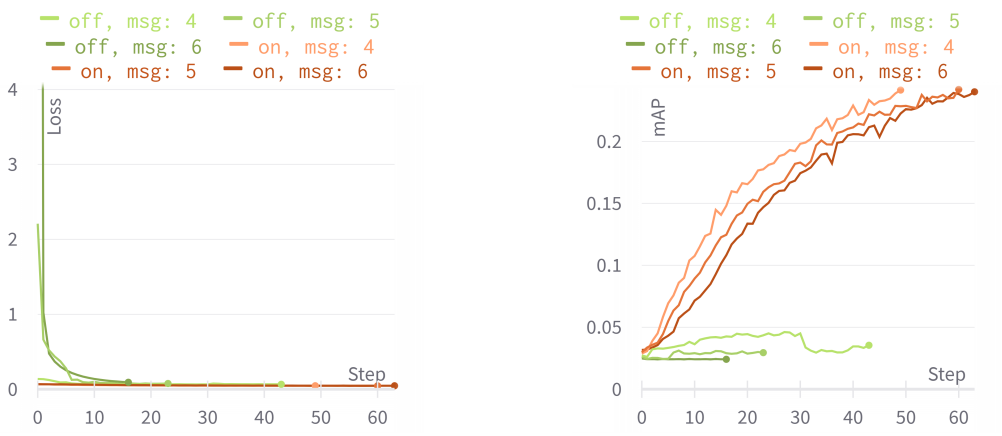
(a) Loss (b) Accuracy

Figure 7: Varying activation function (relu/gelu/sigmoid/hard_tanh) and model width (100/200) for MNIST MLP



(a) Loss (b) Accuracy

Figure 8: Varying batchnorm (on/off) and model depth (3/5/7) for MNIST MLP



(a) Loss (b) Accuracy

Figure 9: Varying layer norm (on/off) and message passing steps (4/5/6) for GNN

Figure 10: Loss and mAP for various GNN model architectures. Varying activation, latent dimensions, hidden dimensions, and message passing steps.