Emergent Behavior in Shallow Neural Networks Trained on Cyclic Labels

199 Research Report

Cash Bowman
Collaborators (discussion and feedback):
Kevin Liu, Guido Montúfar, Michael Murray

November 24, 2025

Abstract

This report follows an ongoing line of empirical results obtained from training a two-layer ReLU network with hinge loss on a toy dataset with cyclic labels. Our goal is to investigate the generalization behavior and training dynamics of the network.

1 Introduction

This study investigates the behavior of a two-layer ReLU network trained on a sinusoidal dataset using hinge loss as the objective function.

1.1 Setup

We train a two-layer ReLU network on a dataset that consists of input-output pairs $(\mathbf{x}_i, y_i)$, where each $\mathbf{x}_i$ is a real number uniformly sampled from the interval $[-0.5, 0.5]$, and $y_i = \text{sgn}(\sin(2\pi * \textit{periods} * \mathbf{x}_i))$. In this paper, we only look at $\textit{periods} = 3$. We model the network using the following architecture:

1. **First layer (hidden layer):**
   - Weight matrix $\mathbf{W}_1 \in \mathbb{R}^{m \times 1}$
   - Bias vector $\mathbf{b}_1 \in \mathbb{R}^m$
   - Preactivation: $\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$
   - Postactivation: $\mathbf{a}_1 = \text{ReLU}(\mathbf{z}_1)$
2. **Second layer (Output layer):**
   - Weight vector $\mathbf{v} \in \mathbb{R}^m$
   - Final output: $y_{\text{pred}} = \mathbf{v}^\top \mathbf{a}_1$

The model is trained using hinge loss to perform binary classification. For each training example, the hinge loss is computed as:

$$L(\mathbf{x}_i, y_i) = \max(0, 1 - y_i y_{\text{pred},i}),$$

where $y_{\text{pred},i}$ is the predicted output for the i-th input $\mathbf{x}_i$, and y_i is the true label. The total loss function is:

$$\mathcal{L}(\mathbf{W}_1, \mathbf{v}, \mathbf{b}_1) = \sum_{i=1}^{N} \max(0, 1 - y_i y_{\text{pred},i}),$$

where N is the number of data points. Note that we do not take mean of loss by dividing by N. Using mean reduction causes issues during training on tasks like this, where we only care about learning points around the decision boundary. Because we don't normalize, we experience unstable gradients at large values of N.

1.2 Optimization

The model is trained using stochastic gradient descent (SGD) with backpropagation. The gradients of the loss function with respect to the model parameters are computed, and the weights and biases are updated in the direction that minimizes the loss. The parameter updates are given by:

$$\mathbf{W}_1 \leftarrow \mathbf{W}_1 - \eta \nabla_{\mathbf{W}_1} \mathcal{L}, \quad \mathbf{v} \leftarrow \mathbf{v} - \eta \nabla_{\mathbf{v}} \mathcal{L}, \quad \mathbf{b}_1 \leftarrow \mathbf{b}_1 - \eta \nabla_{\mathbf{b}_1} \mathcal{L},$$

where η is the learning rate, and the gradients are computed using backpropagation.

2 AGOP Experiments

Our initial experiments aimed to observe grokking, a generalization phenomenon where testing loss converges to zero long after training loss reaches zero. We investigated Average Gradient Outer Product (AGOP) regularization, as presented by Mallinar et al. [4] (2023), hoping it could cause grokking, as the paper suggested that grokking occurred when regularization was present, but often wouldn't in the absence of regularization.

2.1 AGOP Regularization

Following Mallinar et al. (2023), and as defined in Definition 2.1 of the original paper (Mallinar et al., 2023), the Average Gradient Outer Product (AGOP) is defined as:

$$G(f; \{\mathbf{x}_i\}_{i=1}^{n}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial f(\mathbf{x}_i)}{\partial x} \left(\frac{\partial f(\mathbf{x}_i)}{\partial x} \right)^{\top}, \tag{1}$$

where:

- $f(x)$ is the output of the network, $y_{\text{pred}} = \mathbf{v}^{\top} \mathbf{a}_1$,
- $\mathbf{x}_i$ represents the input data points,
- $\frac{\partial f(\mathbf{x}_i)}{\partial x}$ is the Jacobian of $f(\mathbf{x}_i)$ with respect to the input x, capturing the gradient of the network's output with respect to its inputs.

For our two-layer ReLU network:

1. The hidden layer output is $\mathbf{a}_1 = \text{ReLU}(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$.
2. The final output is $y_{\text{pred}} = \mathbf{v}^{\top} \mathbf{a}_1$.

The Jacobian of y_{pred} with respect to x is:

$$\frac{\partial y_{\text{pred}}}{\partial x} = \mathbf{v}^\top \frac{\partial \mathbf{a}_1}{\partial x}, \tag{2}$$

where $\frac{\partial \mathbf{a}_1}{\partial x}$ depends on the ReLU activation.

2.2 Loss Function with AGOP - Trace Penalty

The hinge loss with AGOP regularization, penalized by the trace of the AGOP matrix, is:

$$\mathcal{L}(\mathbf{W}_1, \mathbf{v}, \mathbf{b}_1) = \sum_{i=1}^{N} \max(0, 1 - y_i y_{\text{pred},i}) + \lambda \cdot \text{Tr}\left[G(f; \{\mathbf{x}_i\}_{i=1}^N)\right], \tag{3}$$

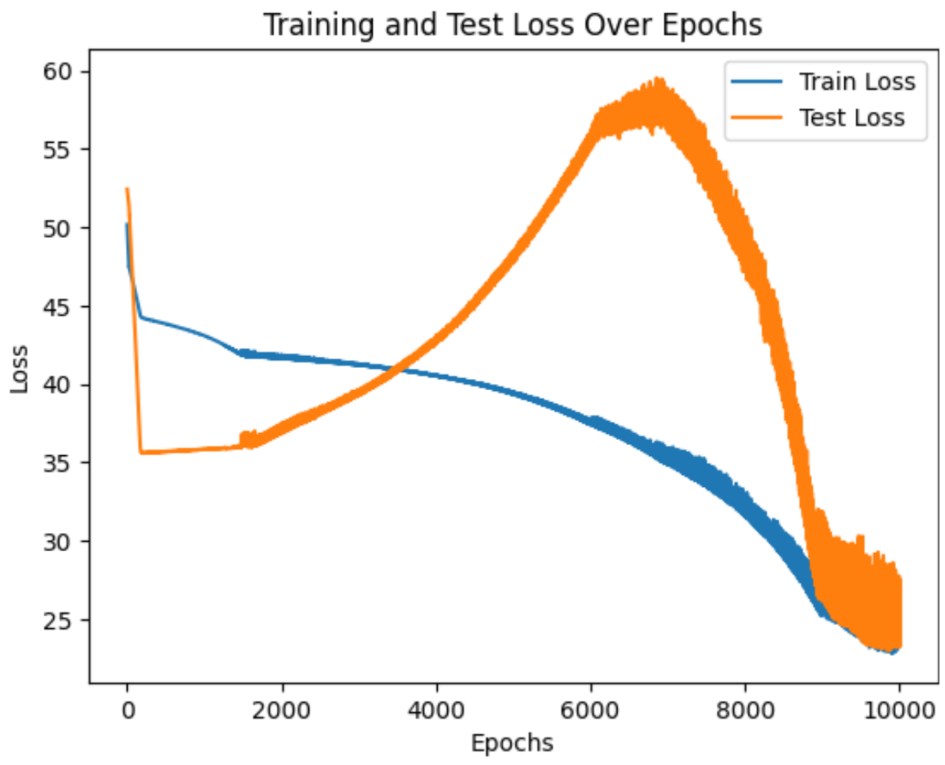
where:

$$G(f; \{\mathbf{x}_i\}_{i=1}^N) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\partial f(\mathbf{x}_i)}{\partial x}\right) \left(\frac{\partial f(\mathbf{x}_i)}{\partial x}\right)^\top, \tag{4}$$

and where $\text{Tr}[A]$ denotes the trace of matrix A, N is the number of data points, and $\lambda > 0$ controls the regularization strength.

2.3 Outcomes

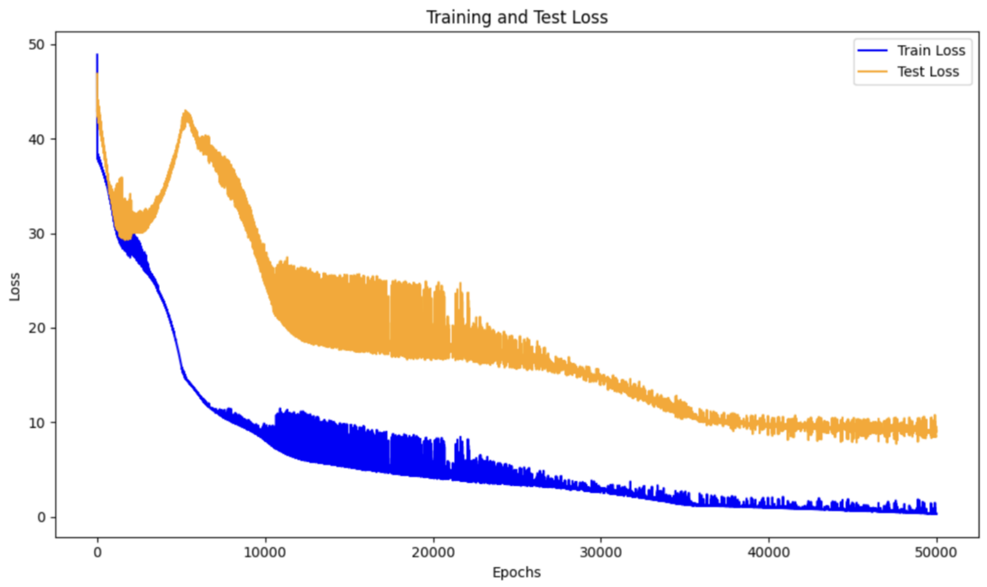
Our initial experiments integrated AGOP regularization with a regularization coefficient $\lambda = 0.001$. We ran multiple models, each with different randomly sampled data and initialized weights. One of our train-test loss curves was particularly striking.



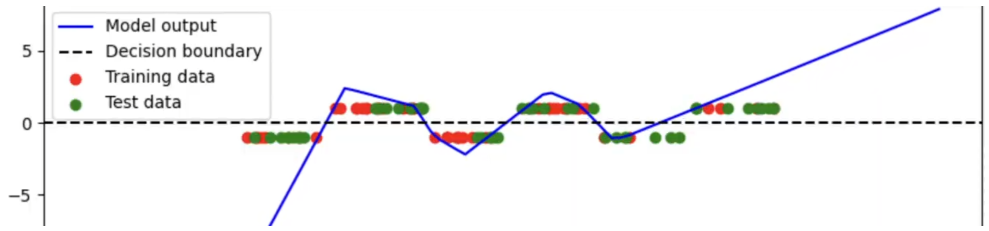
This behavior is very odd, and to get a better grasp of how the model output was changing over training, we recorded model outputs over the epochs to create a video displaying the model's evolution.

3 Model Output Investigation

First, we examined if AGOP regularization was worth further pursuing, and found that it had little affect on the models outcome. In fact, other regularization techniques, like weight decay, also seemed not to prevent the model from having strange runs, such as the one pictured above; so the rest of our model's runs were performed without regularization. To better understand what was causing the odd test loss curves, Dr. Michael Murray proposed I create animations that would display the model outputs over the epochs. Here is a link to the animation of the model that produced the train-test loss curve below.



Below is the model outputs at the end of its training.



The model animation, along with a good look at the sampling of the train and test data, indicates that the odd train-test loss behavior is largely due to under sampling some portions of the periods. In the case above, the train data is under sampled in the two rightmost clusters. While it may not be surprising that under sampling can yield odd training dynamics, it is still interesting how our model can be trained on under-sampled data yet still perform well.

4 Variance Investigation

To understand how under sampling affects the performance of the model, we developed the following experiment.

1. For sample sizes $n = 10 : 200$
2. Train 10 models with randomly initialized weights and randomly sampled training/test data.
3. For epochs $t = 1 : 20,000$
   (a) Compute $L(n, t, s)$, the training loss on data set every 500 epochs.
   (b) Compute $V(n, t)$, the variance of the trained function every 500 epochs.
4. Randomize the initial parameters near 0 and repeat the experiment.

This experimental setup can give us insight in how the data set size, training loss, and variance interact and change under different conditions, including randomized initialization of parameters. The entries of the left heatmap plots the variance summed across all 10 models of a given sample size. The Right heatmap plots the losses summed across all 10 models of a given sample size.

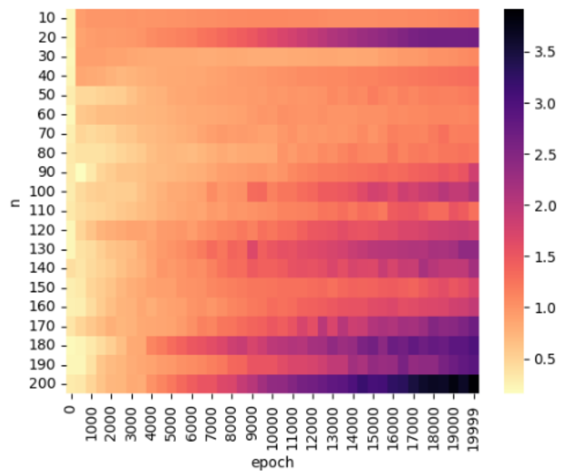


Figure 1: 3 Periods Variance Heatmap (Sample Size n)

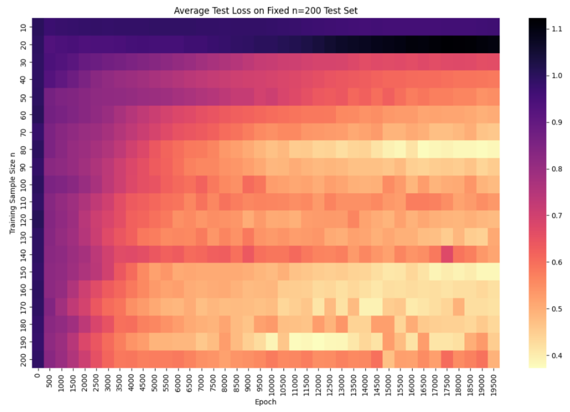


Figure 2: 3 Periods Losses Heatmap

To try and explain the variance of the trajectories, we explored antisymmetric initialization. This experiment employs the antisymmetric initialization (ASI) trick proposed in *A type of generalization error induced by initialization in deep neural networks* by Zhang et al.[10] (2020). To eliminate the negative impact of non-zero initial output, simply initializing to zero fails since $\theta = 0$ leads to a saddle point in the risk surface $R_S(\theta)$. This causes nonlinear training dynamics not captured by NTK. Small initial parameters also vanish the NTK kernel, slowing learning.

To address this, the **AntiSymmetrical Initialization (ASI)** trick zeroes the initial output while preserving the kernel.

Assume a DNN with activation function $\sigma \in C^1(\mathbb{R})$. Let $h_i^{[l]}$ be the pre-activation output of the i-th node in the l-th layer. For $l = 1, \dots, m_L$:

$$h_i^{[l]}(x) = \sigma\left(W_i^{[l]} \cdot h^{[l-1]}(x) + b_i^{[l]}\right)$$

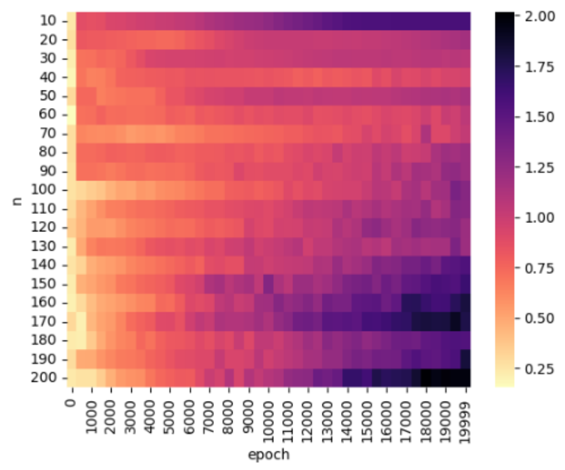


Figure 3: 4 Periods Variance Heatmap (Sample Size n)

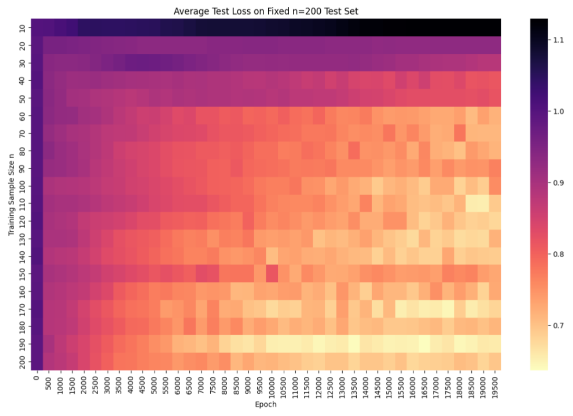


Figure 4: 4 Periods Losses Heatmap

The output layer is:

$$h^{[L]}(x) = W^{[L]}h^{[L-1]} + b^{[L]}$$

After any initialization, denote:

$$h^{[L]}(x, \theta(0)) = \left(W^{[L]}(0), b^{[L]}(0), \ldots, W^{[1]}(0), b^{[1]}(0)\right)$$

The ASI trick defines a new DNN:

$$h_{\text{ASI}}(x, \Theta(t)) = \frac{\sqrt{2}}{2}h^{[L]}(x, \theta(t)) - \frac{\sqrt{2}}{2}h^{[L]}(x, \theta'(t))$$

with $\Theta = (\theta, \theta')$ initialized such that $\theta'(0) = \theta(0)$. The heatmap from the antisymmetric initialization experiment is shown below.

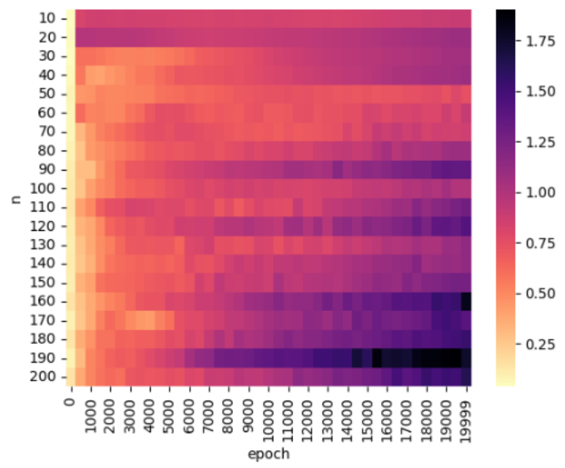


Figure 5: ASI Variance Heatmap (Sample Size n)

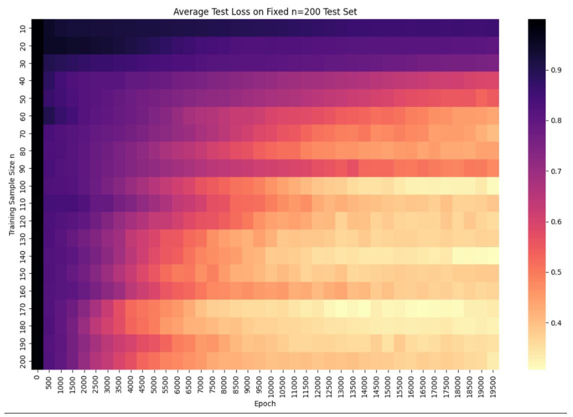


Figure 6: ASI Test Losses Heatmap

As you can see, the results did not differ much from the normal initialization. I believe the reason we observe increasing variance with more training epochs and larger sample sizes is due to the properties of the hinge loss function. Specifically, hinge loss does not normalize with respect to sample size, which can lead to unstable gradients as the dataset grows. Also, since the loss becomes zero once the predictions are beyond the margin, different models can converge to functions with varying amplitudes, resulting in high variance between the learned solutions.

5 Plotting Loss Gradient Projected onto Eigenvectors of NTK at Initialization

We empirically analyze Neural Tangent Kernel (NTK) training dynamics.

Let $\Theta_0 = JJ^\top$ be the NTK at initialization, where $J \in \mathbb{R}^{n\times p}$ is the Jacobian of model outputs with respect to parameters.

The goal of this experiment is to analyze the behavior of a neural network in the Neural Tangent Kernel (NTK) regime under hinge loss by observing how the gradient evolves with respect to the NTK eigenbasis.

We perform the following steps:

1. **Compute the NTK Gram matrix** at initialization:
$$K = JJ^\top, \tag{5}$$
where J is the Jacobian of the model output with respect to its parameters.

2. **Eigendecompose the NTK:**
$$K = \sum_i \lambda_i \xi_i \xi_i^\top, \tag{6}$$
where λ_i are the eigenvalues and ξ_i are the corresponding orthonormal eigenvectors.

3. **Compute the gradient of the loss** at each epoch during training:

   When training with **hinge loss**, the quantity that drives parameter updates is the gradient of the loss with respect to the model outputs:
$$\nabla_{f(x)} \mathcal{L}_{\text{hinge}}(f(x), y), \tag{7}$$
rather than the residuals.

4. **Project the gradient** onto the NTK eigenbasis:
$$\beta_i(t) = \left\langle \nabla_{f(x)} \mathcal{L}_{\text{hinge}}(t), \xi_i \right\rangle, \tag{8}$$
where ξ_i is the i-th eigenvector of the NTK at initialization.

   This projection quantifies how strongly the optimization direction at epoch t aligns with each NTK mode. Since modes with larger λ_i converge faster, tracking $\beta_i(t)$ reveals how different frequency components of the target signal are prioritized by gradient descent under the hinge loss.

5. **Track and plot** how these coefficients $\beta_i(t)$ evolve throughout training to understand the spectral dynamics of learning.

The key objective of this is to see if there is an alignment in the bump in test error with some activity in the NTK projection plot.

To decide how many eigenmodes to include, I examined the spectrum of the NTK at initialization. Let the eigenvalues of the NTK matrix be sorted in descending order:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

Define the cumulative energy captured by the top k eigenvalues as:

$$E(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

Then, the number of modes K is chosen as:

$$K = \max\left(5,\ \min\left\{k \in \mathbb{N} \mid E(k) \geq 0.95\right\}\right)$$

The below image shows the distribution of the NTK eigenvalue values at initialization. This is commonly called plotting the spectrum of the NTK.

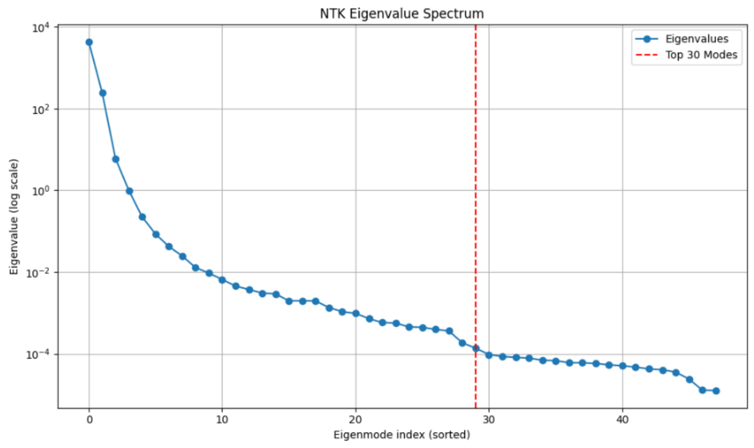


Figure 7: **NTK Eigenspectrum for Width 512.** We plot the eigenvalues of the NTK at initialization to analyze its spectrum. In my experiments, I applied this procedure across all network widths. This threshold was always met within the top 5 eigenvalues, usually the first two, so we consistently plotted 5 modes. Across all widths, the top eigenvalue consistently dominated the spectrum, indicating strong alignment with low-complexity (low-frequency) modes early in training.

In this experiment, we compute and plot $\beta_i(t)$ over time for the top eigenmodes. This helps us understand:

- which modes dominate the learning signal,
- how sparsely the NTK spectrum is utilized,
- and how the optimizer navigates the function space.

The following plots show the test-loss curve and gradient of the loss projected onto the NTK eigenvectors at initialization for different network widths. In each width, we monitor the top five eigenmodes.

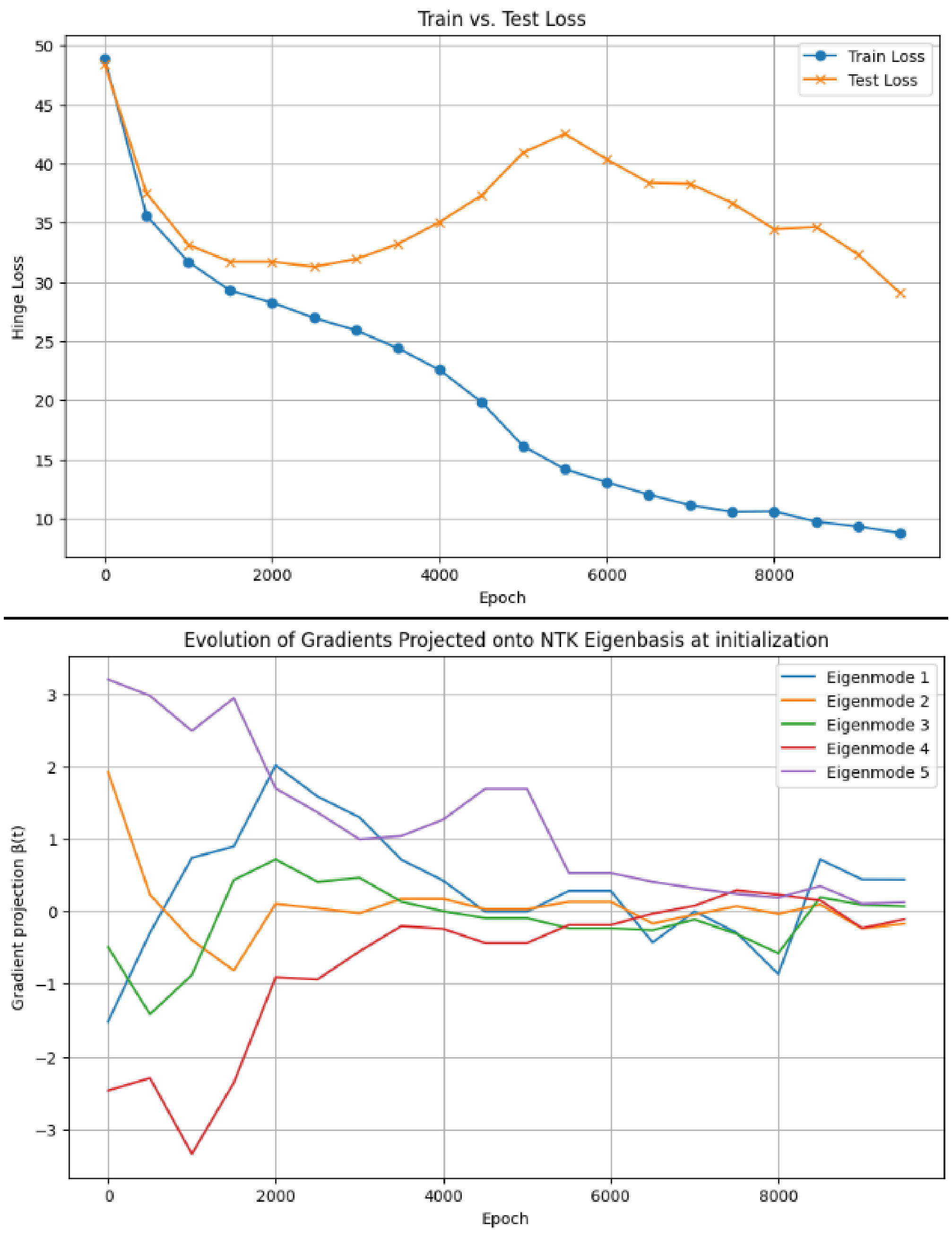


Figure 8: Gradient projections for width 128. Most eigenmodes begin with significant contribution and gradually decay toward zero, with relatively smooth dynamics and no clear mode dominating long term. Notice that right when the test error begins to increase, all of the gradient projections begin to converge towards 0, except eigenmode 5.

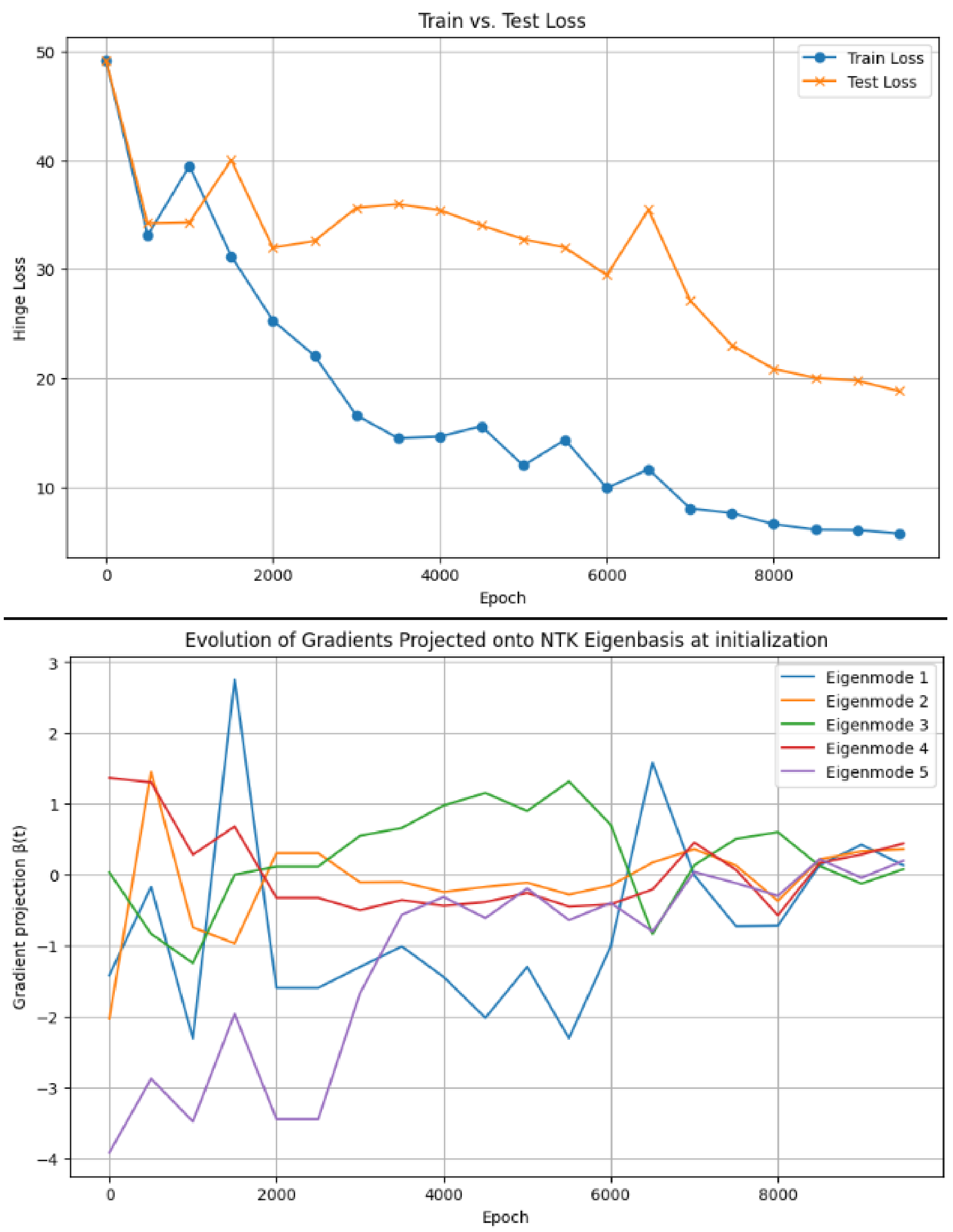


Figure 9: Gradient projections for width 512. Fluctuations are larger early in training, especially for the top two modes. Eigenmodes stabilize near zero after mid-training, with mode 3 contributing longest.

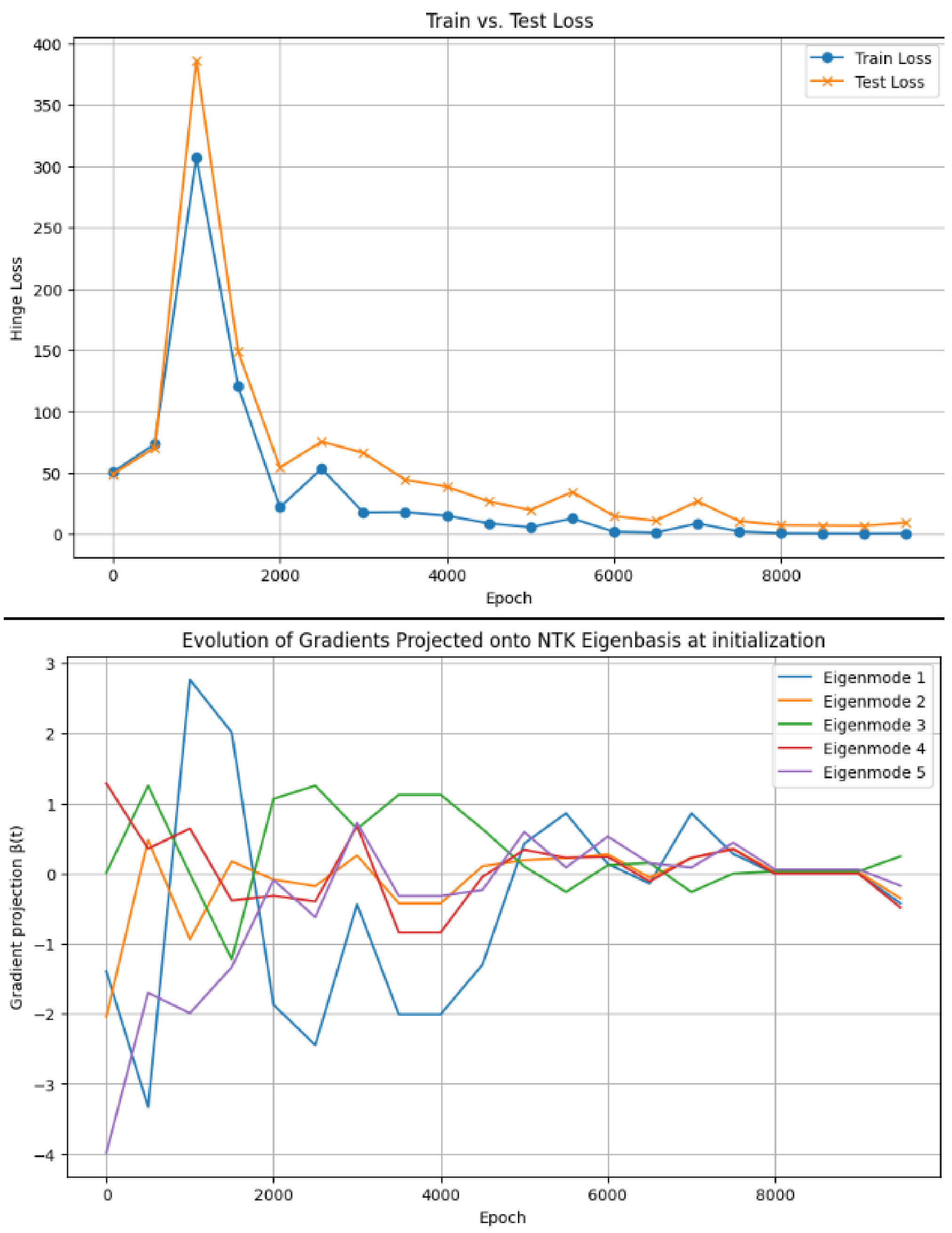


Figure 10: Gradient projections for width 8192. All eigenmodes exhibit sharp early oscillations followed by rapid convergence toward zero. This suggests highly aligned, noise-sensitive dynamics with strong compression in gradient directionality.

6 Next Steps

- Can we think of a framework that would explain the evolution where the train and test errors diverge and then converge? Run ablations on the model to try and isolate the causes of the increase in test-loss. Current hypothesis: This is controlled by an undersampling of clusters.
- Can this be explained in a feature learning perspective? For this, what might be the relevant

features?

- Imbalanced samples. Consider a linear classifier. For this, consider a fixed random embedding of the original data. See if one can use results from the paper of Yiqiao et al.
- Is the average gradient outer product upweighting important features–related to the class imbalance–when the test loss starts to improve? Refer to the paper's experiments on STL-10 dataset [7] (2024).

Some papers that are related to the above questions:

- *Neural tangent kernel: Convergence and generalization in neural networks*
- *Towards understanding the spectral bias of deep learning*
- *Feature learning in infinite-width neural networks*
- *A statistical theory of overfitting for imbalanced classification*
- *Mechanism of feature learning in deep fully connected networks and kernel machines that recursively learn features*

For reproducibility and implementation details, the following notebooks contain code used in this study:

- `Variance_and_Loss_Heatmaps.ipynb` — Variance and test loss visualizations across sample sizes. Also includes ASI experiments.
- `NTK_Tracking_and_Projections.ipynb` — NTK eigenspectrum and projection analysis across network widths. Also includes projecting the residuals onto the NTK eigenbasis, which I ommitted from this report.

References

[1] Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning, 2020.

[2] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020.

[3] Jingyang Lyu, Kangjie Zhou, and Yiqiao Zhong. A statistical theory of overfitting for imbalanced classification, 2025.

[4] Neil Mallinar, Daniel Beaglehole, Libin Zhu, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Emergence in non-neural models: grokking modular arithmetic via average gradient outer product, 2024.

[5] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019.

[6] Amanda Olmin and Fredrik Lindsten. Towards understanding epoch-wise double descent in two-layer linear neural networks, 2024.

[7] Adityanarayanan Radhakrishnan, Daniel Beaglehole, Parthe Pandit, and Mikhail Belkin. Mechanism for feature learning in neural networks and backpropagation-free machine learning models. *Science*, 383(6690):1461–1467, 2024.

[8] Yuqing Wang, Zhenghao Xu, Tuo Zhao, and Molei Tao. Good regularity creates large learning rate implicit biases: edge of stability, balancing, and catapult, 2023.

[9] Greg Yang and Edward J. Hu. Feature learning in infinite-width neural networks, 2022.

[10] Yaoyu Zhang, Zhi-Qin John Xu, Tao Luo, and Zheng Ma. A type of generalization error induced by initialization in deep neural networks, 2020.

[4] [10] [7]