Neural Networks at The Edge of Stability

Matthew Jacobsen
majacobsen@ucsd.edu

Tianhao Wang
tianhaowang@ucsd.edu

Abstract

This report replicates the results of Cohen et al. (2021), which demonstrates that neural networks trained with gradient descent operate in a region called the edge of stability. In this regime, the training loss oscillates in the short-term but decreases overall. We first describe the edge of stability and its causes. We then show its existence empirically for gradient descent, gradient descent with momentum, and RMSProp on a standard neural network.

Code: https://github.com/mfjacobsen/Edge_of_Stability

1 Introduction

Neural networks trained using gradient descent train in an environment known as the edge of stability. It's characterized by a training loss that oscillates in the short-term, but decreases in the long term. The edge of stability is caused by a phenomenon in the training loss function called ”progressive sharpening.” Sharpness is defined as the largest eigenvalue of the Hessian of the loss function. As training progresses, sharpness progressively increases each epoch until it is at or slightly above $2/\eta$, where η is the learning rate. Once it reaches this edge, the previously smooth and stable loss function becomes unstable.

The first paper to describe in detail the edge of stability for full batch gradient descent in neural networks was Cohen et al. (2021). They train a variety of neural networks, including gradient descent with momentum, and show that the edge of stability is widespread in common architectures. Damian, Nichani and Lee (2023) describe the mechanism which prevents the training loss from completely diverging. Once the edge of stability is reached, the third-order term in the loss Taylor expansion shows a negative feedback that counteracts divergence in the sharpest direction, bringing the loss function back to stability.

2 Background

2.1 Progressive Sharpening

As described in Cohen et al. (2021), neural networks have an overwhelming tendency to progressively sharpen during training. Sharpness may decrease very early in training, but this is followed by progressive sharpening, to a threshold of $2/\eta$. Once this threshold is reached, sharpness stays slightly above or below it for the duration of training. However, if the learning rate is small enough, the network may never reach the threshold during training. One exception to progressive sharpening is networks with cross-entropy loss functions. These exhibit progressive sharpening to the threshold, but a decrease late in training. This is because cross-entropy loss pushed the model to increase the margins of correctly classified observations later in training, which decreases the second derivatives and therefore sharpness.

2.2 Edge of Stability

Once the neural network has sharpened to the threshold of $2/\eta$, it enters a regime called the edge of stability. This phenomenon is characterized by short-term oscillations and long-term decrease of the training loss. Along with the oscillation of the training loss is the oscillation of sharpness around the threshold.

The mechanics for why gradient descent does not diverge after reaching the sharpness threshold are explored in Damian, Nichani and Lee (2023). After sharpness exceeds the $2/\eta$, it theoretically should start to diverge along the corresponding eigen-direction. At this point, the quadratic approximation of the loss function is no longer valid and a third-order Taylor expansion is needed. They find that the cubic term in this expansion is proportional to the gradient of the sharpness. As the loss function continues to diverge, this cubic term comes to dominate. Since the gradient descent update formula pushes the loss in the negative direction, it pushes the updates into regions with lower curvature. That is, the network implicitly pushes the updates into regions with lower sharpness.

2.3 Quadratic Approximations of Neural Networks

For a neural network with d parameters and weight vector $w^{(t)} \in \mathbb{R}^d$ at time t with a loss function $L(w)$ and learning rate η, the update rule for gradient descent is:

$$w^{(t+1)} = w^{(t)} - \eta \nabla L(w^{(t)})$$

The theory of stability and the $2/\eta$ sharpness threshold comes from gradient descent performed on quadratic functions. Neural network loss functions are not quadratic. In fact, their Hessians are non-convex and indefinite. However, by taking a second-order Taylor expansion around the current iterate of the weights $w^{(t)}$, the loss is locally approximated by a quadratic with a Hessian $H_t = \nabla^2 L(w^{(t)})$.

3 Experimental Design

Cohen et al. (2021) and this report use a 5,000 image subset of the CIFAR-10 dataset. CIFAR-10 consists of 32×32 color images of ten object classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The subset maintains the original data set class distribution while reducing its overall size. Using this smaller sample enables faster experimentation and makes full-batch gradient descent more feasible. The dataset is included in the pytorch python library.

The neural network architecture used in this report is as follows. A fully-connected neural network with an input size of 3072 (CIFAR-10 is 32x32x3), followed by two hidden layers of 200 neurons each, with an output layer of 10 classes. The total number of parameters in the network is 656,810. Activation function used was tanh, with Xavier uniform weight initialization. Input data was 0-mean normalized before training and networks were trained to completion (at least 99% training accuracy).

4 Stability Analysis

We will now analyze different optimizers and their quadratic proofs for edge of stability. Following each analysis, we show that the edge of stability is reached at the theoretical sharpness limit on our experimental neural network

4.1 Gradient Descent

We will begin with an analysis on stability of gradient descent on convex quadratic functions. For $x \in \mathbb{R}^d$, a quadratic function $f(x) = \frac{1}{2}x^\top Ax + b^\top x$, and a learning rate η, the update rule for gradient descent is:

$$x^{(t+1)} = x^{(t)} - \eta \nabla f(x^{(t)})$$

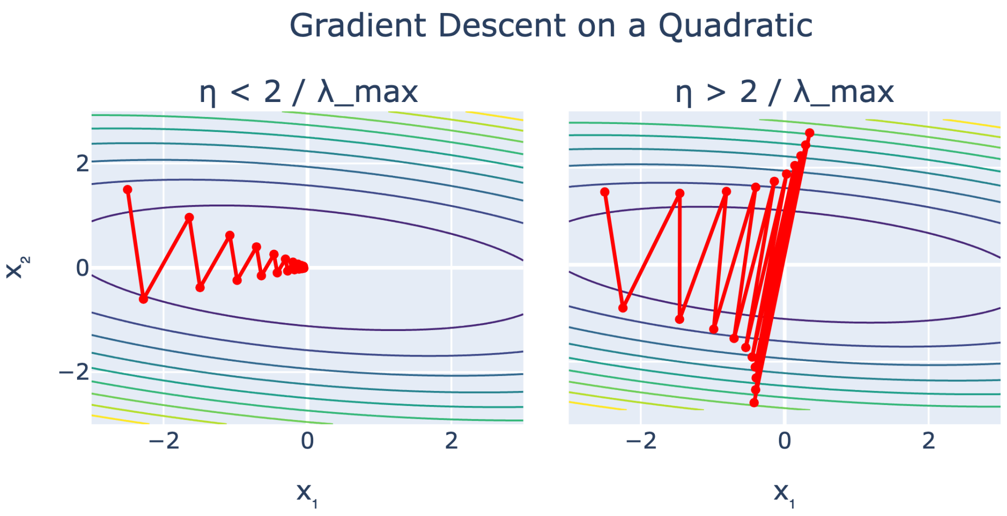


Figure 1: Gradient descent on a quadratic with learning a learning rate below the stability threshold (left) and above the stability threshold (right). Only one of the eigenvalues is above the threshold, resulting in divergence along the corresponding eigenvector, with convergence along the second eigenvector.

The minimizer of f is x^*, where $\nabla f(x^*) = Ax^* + b = 0$. We define the error from any point $x^{(t)}$ to the minimizer as $e^{(t)} = x^{(t)} - x^*$. With this definition, we redefine the update as an error from the minimizer:

$$\begin{aligned}
x^{(t+1)} &= x^{(t)} - \eta \nabla f(x^{(t)}) \\
e^{(t+1)} &= e^{(t)} - \eta \nabla f(e^{(t)} + x^*) \\
&= e^{(t)} - \eta \left(A(e^{(t)} + x^*) + b\right) \\
&= e^{(t)} - \eta \left(Ae^{(t)} + Ax^* + b\right) \\
&= e^{(t)} - \eta Ae^{(t)} \\
e^{(t+1)} &= (I - \eta A)e^{(t)}
\end{aligned}$$

Since A is symmetric, we can diagonalize it as $A = Q\Lambda Q^\top$. Then, we will redefine our error term by rotating it with the eigenbasis Q: $g^{(t)} = Q^\top e^{(t)}$. Diagonalizing A and re-basing the error gives us our final form:

$$
\begin{aligned}
e^{(t+1)} &= (I - \eta Q \Lambda Q^\top) e^{(t)} \\
Q g^{(t+1)} &= (I - \eta Q \Lambda Q^\top) Q g^{(t)} \\
g^{(t+1)} &= (I - \eta \Lambda) g^{(t)}
\end{aligned}
$$

Here, we note that since f is convex, all eigenvalues of A are greater than 0, that is $\forall\, i \in \{1, 2, ...d\}$, $\lambda_i \geq 0$. Since Λ is a diagonal matrix of λ_i's, we have decoupled the gradient descent update into the eigen-directions of A. Since we aim to minimize $g^{(t)}$, each successive iteration should be smaller than the last. That is $\forall\; i$, $|1 - \eta\lambda_i| < 1$, or equivalently, $0 < \eta\lambda_i < 2$. Since all $\lambda_i \geq 0$, we can ignore the lower bound, and we have $\eta\lambda_i < 2$.

Therefore, for gradient descent to remain stable and not diverge on the quadratic function f, the learning rate η must be chosen as $\eta < \frac{2}{\lambda_{max}}$ where $\lambda_{max} = \arg\max_i \lambda_i$. The eigenvalue λ_{max} of A is called the sharpness of A. We can alternatively say that if the sharpness of A exceeds $\frac{2}{\eta}$, then gradient descent will diverge.

By Section 2.1, the same stability analysis applies locally to a neural network. Figure 2 shows that just like the quadratic case, if any eigenvalues of the Hessian H_t exceed $2/\eta$, then gradient descent will become locally unstable along the corresponding eigen-direction.

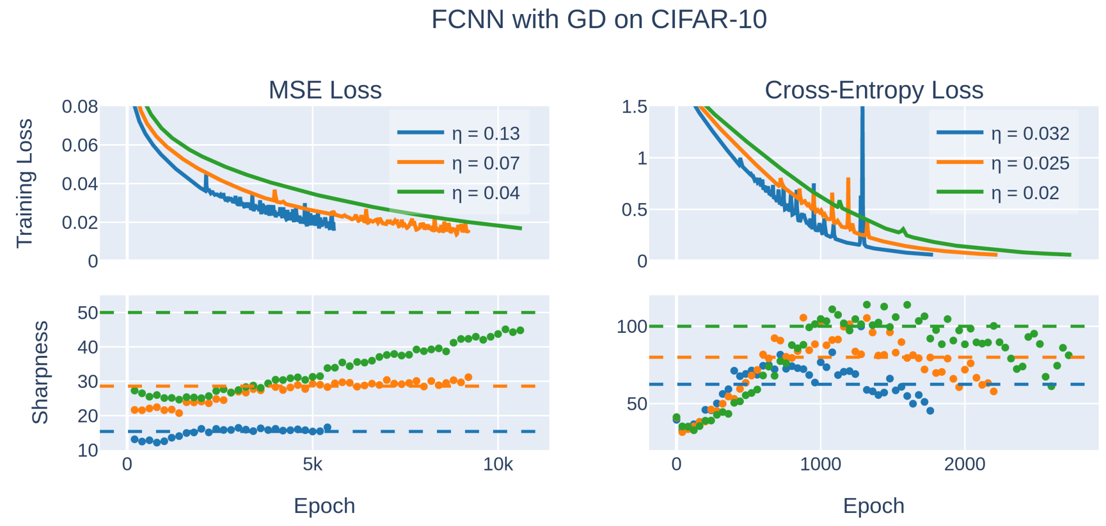


Figure 2: A fully connected neural network trained on CIFAR-10 with 2 hidden layers of 200 neurons each and tanh activations. Progressive sharpening is observed in the bottom plots to the stability threshold 2 / η, at which point the training loss destabilizes. For low learning rates, the network remains stable throughout training. Observe that for cross-entropy loss, sharpness drops in at the end of training.

4.2 Gradient Descent with Momentum

Gradient descent with Polyak momentum is common method for speeding convergence of gradient descent and is also used in more advanced optimizers like Adam. It incorporates a a velocity term that accumulates a decaying average of past gradients, controlled by a momentum constant β.

We will analyze this method for the quadratic case, as outlined in Cohen et al. (2021). For $x \in \mathbb{R}^d$, a quadratic function $f(x) = \frac{1}{2}x^\top Ax + b^\top x$, and a learning rate η, the update rule for gradient descent with Polyak momentum is:

$$x^{(t+1)} = x^{(t)} - \eta \nabla f(x^{(t)}) + \beta(x^{(t)} - x^{(t-1)}),$$

The minimizer of f is x^*, where $\nabla f(x^*) = Ax^* + b = 0$. We define the error from any point $x^{(t)}$ to the minimizer as $e^{(t)} = x^{(t)} - x^*$. With this definition, we redefine the update as an error from the minimizer:

$$\begin{aligned}
x^{(t+1)} &= x^{(t)} - \eta \nabla f(x^{(t)}) + \beta(x^{(t)} - x^{(t-1)}) \\
e^{(t+1)} &= e^{(t)} - \eta \nabla f(e^{(t)} + x^*) + \beta(e^{(t)} - e^{(t-1)}) \\
&= e^{(t)} - \eta \left(A(e^{(t)} + x^*) + b\right) + \beta(e^{(t)} - e^{(t-1)}) \\
&= e^{(t)} - \eta \left(Ae^{(t)} + Ax^* + b\right) + \beta(e^{(t)} - e^{(t-1)}) \\
&= e^{(t)} - \eta Ae^{(t)} + \beta(e^{(t)} - e^{(t-1)}) \\
e^{(t+1)} &= (I - \eta A + \beta I)e^{(t)} - \beta e^{(t-1)}
\end{aligned}$$

Since A is symmetric, we can diagonalize it as $A = Q\Lambda Q^\top$. Then, we will redefine our error term by rotating it with the eigenbasis Q: $g^{(t)} = Q^\top e^{(t)}$. Diagonalizing A and re-basing the error gives us a linear homogeneous 2nd order recurrence relation:

$$\begin{aligned}
e^{(t+1)} &= (I - \eta Q\Lambda Q^\top + \beta I)e^{(t)} - \beta e^{(t-1)} \\
Qg^{(t+1)} &= (I - \eta Q\Lambda Q^\top + \beta I)Qg^{(t)} - \beta Qg^{(t-1)} \\
g^{(t+1)} &= (I - \eta \Lambda + \beta I)g^{(t)} - \beta g^{(t-1)}
\end{aligned}$$

Notice here that Λ is a diagonal matrix of eigenvalues. We have again decoupled the update into it's eigen-directions and can evaluate each eigen-direction independently for convergence. If for any eigenvalue λ_i the recurrence diverges, then gradient descent with momentum will diverge along the corresponding eigenvector. That is, we evaluate the following recurrence for all $i \in [1, d]$:

$$g_i^{(t+1)} = (1 - \eta \lambda_i + \beta)g_i^{(t)} - \beta g_i^{(t-1)}$$

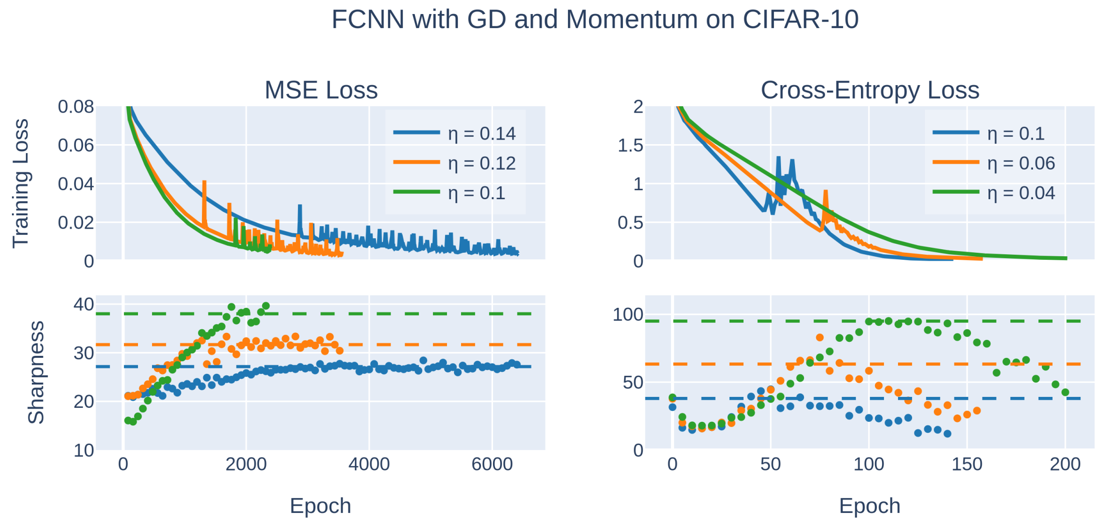
For a 2nd order recurrence of the form $r^{(t+1)} = ar^{(t)} - br^{(t-1)}$, convergence is determined by satisfaction of the Schur stability conditions:

$$1 + a + b > 0 \quad (1)$$
$$1 - a + b > 0 \quad (2)$$
$$1 - b > 0 \quad (3)$$

Applying these conditions for $a = 1 - \eta\lambda + \beta$ and $b = \beta$ gives us stability when $0 < \lambda < 2(1+\beta)/\eta$. Also note that the momentum term β must be less than 1. Therefore, stability of the method is governed by the largest eigenvalue, that is, the system is stable when:

$$\lambda_{max} < \frac{2(1+\beta)}{\eta}.$$

The same argument given for gradient descent on neural networks applies to gradient descent with momentum on neural networks. Although not quadratic, we approximate the Hessian as locally quadratic at time t. The method will diverge if the max eigenvalue of the Hessian exceeds $2(1+\beta)/\eta$.



is reached. A decrease in sharpness for cross-entropy loss is observed later in training.

Figure 3: Progressive sharpening is observed in both loss functions, with instability after the sharpness threshold of $\frac{2(1+\beta)}{}$

4.3 RMSProp

RMSProp (Root Mean Squared Propagation) is an optimization method that adaptively scales the coordinates of the gradient with an exponential moving average of past gradients.

It dampens large changes in the gradient, while increasing the update size in flat directions. Cohen et al. (2025) discusses that the edge of stability is determined by the sharpness of the ”effective Hessian”. That is, the Hessian that after it’s preconditioning.

This is an analysis for the quadratic case. For $x \in \mathbb{R}^d$, a quadratic function $f(x) = \frac{1}{2}x^\top Ax + b^\top x$, and a learning rate η, the update rule for RMSProp is:

$$
\begin{aligned}
v^{(t)} &= \beta v^{(t-1)} + (1-\beta)\nabla f(x)^{\odot 2} \\
x^{(t+1)} &= x^{(t)} - \eta \frac{\nabla f(x^{(t)})}{\sqrt{v^{(t)} + \epsilon}}
\end{aligned}
$$

Note that ϵ is a small constant used to prevent division by zero and ensure numerical stability. The minimizer of f is x^*, where $\nabla f(x^*) = Ax^* + b = 0$. We define the element-wise division in second term of the update as a diagonal-matrix vector product $\frac{\nabla f(x^{(t)})}{\sqrt{v^{(t)}+\epsilon}} = D\nabla f(x^{(t)})$. Near the minimizer, $v \to 0$ and $D \to \epsilon^{\frac{1}{2}} I$, so we treat D as constant across updates. We define the error from any point $x^{(t)}$ to the minimizer as $e^{(t)} = x^{(t)} - x^*$. With this definition, we redefine the update as an error from the minimizer:

$$
\begin{aligned}
x^{(t+1)} &= x^{(t)} - \eta D \nabla f(x^{(t)}) \\
e^{(t+1)} &= e^{(t)} - \eta D \nabla f(e^{(t)} + x^*) \\
&= e^{(t)} - \eta D \left(A(e^{(t)} + x^*) + b\right) \\
&= e^{(t)} - \eta D \left(Ae^{(t)} + Ax^* + b\right) \\
&= e^{(t)} - \eta DAe^{(t)} \\
e^{(t+1)} &= (I - \eta DA)e^{(t)}
\end{aligned}
$$

In order to maintain a symmetric update matrix, we set $u^{(t)} = D^{\frac{1}{2}} e^t$ and define the effective Hessian of the loss as $E = D^{\frac{1}{2}} A D^{\frac{1}{2}}$:

$$
\begin{aligned}
D^{\frac{1}{2}} e^{(t+1)} &= (I - \eta DA) D^{\frac{1}{2}} e^{(t)} \\
u^{(t+1)} &= (I - \eta D^{\frac{1}{2}} A D^{\frac{1}{2}}) u^{(t)} \\
u^{(t+1)} &= (I - \eta E) u^{(t)}
\end{aligned}
$$

Since E is symmetric, we can diagonalize it as $E = Q\Lambda Q^\top$. Then, we will redefine our error term by rotating it with the eigenbasis Q: $g^{(t)} = Q^\top u^{(t)}$. Diagonalizing E and re-basing the error gives us our final form:

$$
\begin{aligned}
u^{(t+1)} &= (I - \eta Q\Lambda Q^\top) u^{(t)} \\
Qg^{(t+1)} &= (I - \eta Q\Lambda Q^\top) Qg^{(t)} \\
g^{(t+1)} &= (I - \eta \Lambda) g^{(t)}
\end{aligned}
$$

Our update is decoupled along the eigen-directions of E, the effective Hessian after the adaptive preconditioner. Therefore, stability is governed by the largest eigenvalue of E:

$$\lambda_{max} < \frac{2}{\eta}$$

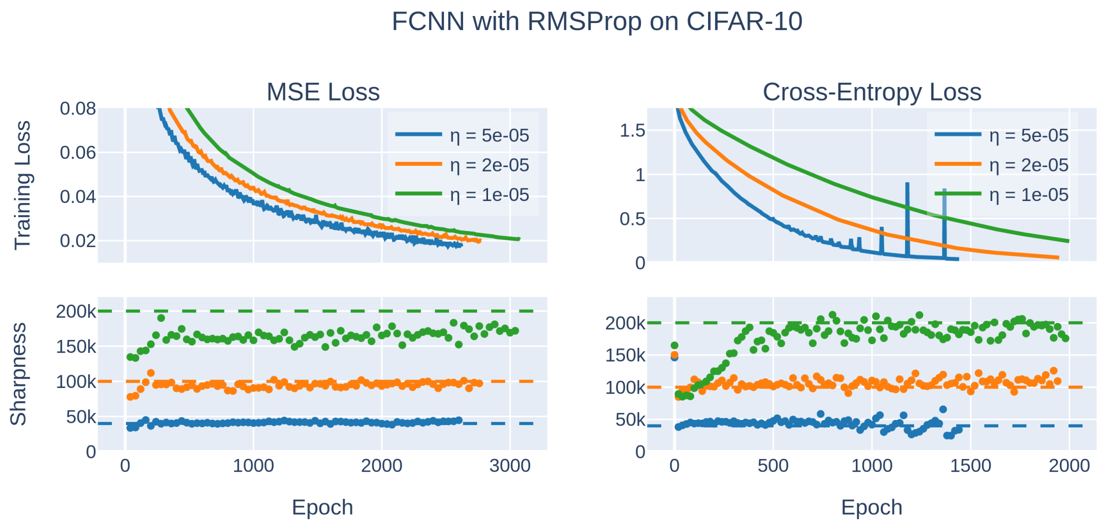


Figure 4: For RMSProp, sharpening of the effective Hessian occurs rapidly with MSE loss functions and the loss is instable stable with very small oscillations. For Cross-Entropy loss, instability manifests as a few very large oscillations.

5 Conclusion

This report reproduces the findings of Cohen et al. (2021) and demonstrates that neural networks trained with full-batch optimizers consistently enter the edge of stability. For gradient descent, gradient descent with momentum, and RMSProp, the sharpness of the loss increases until it reaches the respective stability threshold predicted by quadratic theory. Once this threshold is reached, training exhibits a pattern of short-term oscillations with long-term loss reduction.

Across all optimizers, the empirical sharpness tracks the theoretical limit. $2/\eta$ for gradient descent and RMSProp, and $2(1+\beta)/\eta$ for momentum. This finding confirms that local quadratic approximations accurately predict when instability begins, even though neural network loss surfaces are non-quadratic and non-convex. The experiments also reproduce the known deviation for cross-entropy loss. In this case, sharpness eventually decreases late in training, consistent with margin growth and the resulting reduction in curvature.

Overall, these results support the view that the edge of stability is a reproducible phenomenon in deep learning optimization.

References

Cohen, Jeremy, Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. 2021. "Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability." *CoRR* abs/2103.00065. [Link]

Cohen, Jeremy M., Alex Damian, Ameet Talwalkar, J. Zico Kolter, and Jason D. Lee. 2025. "Understanding Optimization in Deep Learning with Central Flows." [Link]

Damian, Alex, Eshaan Nichani, and Jason D. Lee. 2023. "Self-Stabilization: The Implicit Bias of Gradient Descent at the Edge of Stability." [Link]