Arnold's Bifurcation Principle Applied to Large-Scale AI Training: Real-Time Hardware Prediction of Loss Divergence Using Hessian-Free Eigenvalue Tracking

Prakash Vaithyanathan, M.Sc. (Physics)
Science Teacher, Chennai, India
pvaithyanathan@gmail.com*

September 2026

Abstract

When a very large AI model is trained, the training sometimes breaks on its own. The loss suddenly jumps up, the numbers inside the model become too large or too small to handle correctly, and the run has to be stopped and restarted from an older saved copy. This is called training instability, and on a large training run it can waste a very large amount of computer time and money. Earlier work in machine learning has shown that this kind of breakdown is closely tied to a single number: the sharpness of the loss surface, which is the largest eigenvalue of the Hessian matrix of the loss function. When this number grows too large compared to the learning rate, the training stops behaving smoothly. The problem is that for a model with a billion or more parameters, the full Hessian matrix cannot be built or stored by any real machine: it would need far more numbers than there are atoms within reach. We show that this is not actually needed. A small hardware block can track the same leading eigenvalue using only the gradient vectors that the training hardware is already computing at every step, through a standard numerical method called power iteration, combined with the well-known fact that a Hessian-vector product can be estimated from the difference between two gradients without ever building the Hessian itself. We also show that watching a small, carefully chosen part of the model, rather than the whole thing, is often enough to catch the problem early. We describe a domain-blind hardware controller that watches this shrinking safety margin every single step and estimates how many steps remain before the run is expected to become unstable, so that a corrective action can be taken before the run actually fails rather than after.

Keywords: Arnold's Bifurcation Principle, Edge of Stability, Training Instability, Hessian-Free Tracking, Power Iteration, Hardware Controller, Layer-Local Probing, Adaptive Optimizers.

1 System Model

Let $\vec{\theta}(t) = [\theta_1, \ldots, \theta_N]^T$ be the parameter vector of the model being trained at training step t, where N can be in the billions. Training updates the parameters using gradient descent (or a close variant of it):

$$\theta(t+1) = \theta(t) - \eta\, g(t), \qquad g(t) = \nabla C(\theta(t))$$

where C is the loss function and η is the learning rate.

*Patent Pending.

The local curvature of the loss surface at $\theta(t)$ is described by the Hessian matrix $H(t) = \nabla^2 C(\theta(t))$. Its largest eigenvalue,

$$S(t) = \lambda_{\max}\big(H(t)\big),$$

is usually called the *sharpness* of the loss surface. Earlier published work (Cohen et al., 2021 [1]) showed that gradient descent tends to push $S(t)$ upward during training (a process called *progressive sharpening*) until it reaches a value close to $2/\eta$. Past this point, training does not smoothly converge; instead it enters a regime where the loss can suddenly spike and, in the worst case, the run diverges completely.

We define the *stability margin*

$$m(t) = \frac{2}{\eta} - S(t).$$

While $m(t) > 0$, the run is in a numerically stable regime. As sharpness rises, $m(t)$ shrinks toward zero, and the point $m(t) = 0$ marks the onset of the unstable regime.

We are explicit about what is, and is not, being claimed here. The link between sharpness and training instability is not our discovery; it is one of the more active areas of current deep learning research [1, 2]. What we propose is a way to track this specific quantity, in real time, inside dedicated hardware, at a scale where computing the full Hessian is physically impossible.

2 Eigenvalue Criterion

Exactly as in the general form of Arnold's Bifurcation Principle, we track not just the margin $m(t)$ but its velocity,

$$v_m(t) = \frac{d}{dt} m(t), \qquad T_{\text{instab}}(t) = -\frac{m(t)}{v_m(t)},$$

which gives an estimated number of steps remaining before the margin reaches zero. This is the same eigenvalue-velocity construction used throughout the Arnold's Bifurcation Principle framework, here applied to a margin that is shrinking toward an instability boundary from above, rather than a curvature that is shrinking toward a stall from below.

3 Hessian-Free Tracking via Power Iteration

The obstacle to using this idea at scale is that computing $S(t)$ in the ordinary way requires the full Hessian matrix $H(t)$, which for a billion-parameter model has on the order of 10^{18} entries. No hardware register, and no realistic amount of on-chip memory, can hold this. The construction below avoids ever forming $H(t)$.

3.1 Hessian-vector products come for free from the training itself

A Hessian-vector product $H(t)\, s$ can be estimated, for a small step s, directly from the difference of two gradients:

$$H(t)\, s \;\approx\; \nabla C(\theta(t) + s) - \nabla C(\theta(t)).$$

This is not a new observation; it is the same secant relationship used inside quasi-Newton optimizers such as BFGS. What is useful for our purpose is that gradient descent already produces exactly this pair of quantities on its own, at every step, at no extra computational cost: the actual step already taken,

$$s(t) = \theta(t+1) - \theta(t) = -\eta\, g(t),$$

and the actual next gradient, $g(t+1)$. Their difference,

$$y(t) = g(t+1) - g(t) \;\approx\; H(t)\, s(t),$$

is a genuine Hessian-vector product along the direction the optimizer is already moving in. No additional forward or backward pass through the model is required to obtain it; both $g(t)$ and $g(t+1)$ are already being computed and streamed by the training hardware as an ordinary part of training.

3.2 Recovering the leading eigenvalue by power iteration

A single pair $(s(t), y(t))$ only gives curvature information along one direction, which is not necessarily the direction of largest curvature. To recover $S(t) = \lambda_{\max}(H(t))$, the hardware controller keeps a small running *tracking vector* $u(t)$, restricted to the same size as the model's step vector, and repeatedly folds each new secant pair into it in the manner of power iteration: the tracking vector is nudged toward whatever direction is currently showing the fastest curvature growth, and is renormalized after each step to prevent overflow. Because this is exactly the mechanism behind power iteration, $u(t)$ converges toward the dominant eigenvector of $H(t)$ over a modest number of steps, and the growth rate of $\|H(t)\, u(t)\|$ relative to $\|u(t)\|$ gives a running estimate of $S(t)$. The controller therefore never needs to build, store, or invert any matrix; it only ever holds and updates one vector.

Because progressive sharpening unfolds gradually, over hundreds or thousands of training steps, the Hessian generally changes slowly compared to how often the controller updates its tracking vector, which happens every single step. In most of training, this gives power iteration enough time to keep pace with the slowly shifting direction of largest curvature. The one caveat worth stating plainly is that power iteration's convergence speed depends on how well separated the largest eigenvalue is from the next one down, and published studies of neural network Hessians have found that the top few eigenvalues can sit close together rather than being cleanly separated [3] – often close to the point where early warning matters most. The response-latency threshold in Section 5 should be chosen with this in mind, rather than assuming worst-case tracking speed is never an issue.

4 Layer-Local Probing

It is worth being explicit about why the idea in Section 3 is still not enough on its own at billion-parameter scale. The secant trick removes the need to ever build the full Hessian matrix, but the tracking vector $u(t)$ described there is, by default, still the same size as the whole parameter vector $\theta(t)$. Updating and renormalizing a vector with a billion or more entries, every single training step, inside a small piece of dedicated hardware, is still far more work and far more on-chip memory than is realistic. Removing the Hessian matrix and shrinking the tracking vector are therefore two separate, both-necessary steps, not one idea and an optional extra: Section 3 removes the matrix, and this section removes the need for the tracking vector to ever span the whole model.In large models, instability is not usually spread evenly across all parameters. Several published studies report that early warning signs concentrate in specific parts of the network, such as attention logits or the output layers, well before the rest of the model shows any sign of trouble [6]. We use this to shrink the tracking vector further: instead of tracking curvature over the full parameter vector, the controller can be wired to only the gradient stream of a chosen subset of layers, for example the final attention heads or output embedding, reducing the tracking vector from billions of entries down to

a few thousand. This is closer to a doctor listening with a stethoscope at a few known trouble spots than to monitoring the entire body at once. Which layers to listen to is a configuration choice, not a fixed property of the hardware; the controller itself remains domain-blind. A reasonable default choice of which layers to probe is the very first and the very last layers of the model: the input embedding, which is the first point where raw external data enters the network, and the output embedding together with the final attention heads, which is the last point before the network's output leaves it. These are the two places most directly in contact with the outside world, rather than with the model's own internal representations, and there is already published evidence pointing at both ends. Wortsman et al. (2023) [7] found that the largest attention logits, an early sign of instability, typically appear in the first attention block rather than in the middle of the network. Separately, output logit divergence in the model's final output layer has been identified as its own distinct, recurring cause of training failure [8]. Probing both ends, rather than the middle, is therefore a reasonable and evidence-backed default, while remaining a configuration choice rather than a fixed property of the hardware.

5 Proposed Hardware Controller and Corrective Action

1. The training hardware (GPU, TPU, or similar accelerator) is already computing a gradient vector $g(t)$ at every training step. A side channel taps only the components of this stream belonging to the chosen probe layers and routes them to the controller; the main training computation is not slowed down or altered.

2. The controller keeps the small tracking vector $u(t)$ described above and updates it every step using the secant pair $(s(t), y(t))$ restricted to the probed layers, entirely in local registers.

3. Each cycle, the controller estimates $S(t)$ from the growth of $u(t)$, computes the margin $m(t) = 2/\eta - S(t)$, its velocity $v_m(t)$, and the estimated time to instability $T_{\text{instab}}(t)$.

4. When $T_{\text{instab}}(t)$ falls below a configured response-latency threshold, the controller raises an interrupt to the training software. The corrective action itself is a policy choice, not fixed by the hardware: it may be a temporary cut to the learning rate, gradient clipping, or triggering an early checkpoint, applied before the loss actually spikes rather than after.

The construction above is shown for plain gradient descent for clarity, but large-scale training almost always uses an adaptive optimizer such as AdamW. Because the tracking vector is built from the optimizer's actual step and actual next gradient, rather than from an idealized update rule, it already reflects whatever step the optimizer really took. What changes for an adaptive optimizer is not simply the value of η: the relevant curvature becomes the leading eigenvalue of a preconditioned Hessian rather than the raw loss Hessian, and the threshold itself is different from $2/\eta$. For Adam with $\beta_1 = 0.9$, published work has found this threshold to be close to $38/\eta$ [4], not $2/\eta$. The margin $m(t)$ and threshold used by the controller should therefore be set according to whichever optimizer is actually in use, rather than assumed to carry over unchanged from plain gradient descent; published work has also found that adaptive optimizers behave differently once near this threshold, so this should be treated as a configuration detail to validate for each optimizer, not a settled equivalence.

6 Distinction from Prior Art

The idea that loss-surface sharpness predicts training instability is well established and actively studied [1, 2]. Estimating a neural network's leading Hessian eigenvalue using Hessian-vector products and power iteration, rather than forming the full Hessian, is also an established numerical technique in machine learning research [3, 5]. Monitoring gradient norms, activation sizes, or related quantities as early indicators of instability has likewise been studied in several recent papers [6].

All of the above, as far as we are aware, are carried out in software: as part of the training loop itself, as an added offline analysis step, or as a periodic diagnostic that requires pausing or extending the training computation. None of them proposes a dedicated piece of hardware, sitting beside the main compute path, that taps an already-existing gradient stream at zero added cost, tracks the velocity of the resulting eigenvalue estimate every single cycle, and triggers an automatic corrective action before the loss spike occurs rather than after it is observed. The novelty claimed here is narrow and specific to this real-time, always-on, in-hardware implementation and its velocity-based early-warning trigger, not to the underlying mathematics of sharpness or of Hessian-free eigenvalue estimation, both of which are already known.

7 Glossary of Terms and Symbols

- **Sharpness** (S) — the largest eigenvalue of the loss function's Hessian matrix; a measure of how curved the loss surface is at the current point.
- **Edge of stability** — the regime, described in recent machine learning research, where the sharpness rises until it sits near $2/\eta$ and training stops behaving smoothly.
- **Hessian-vector product** — the result of multiplying the Hessian matrix by a vector, which can be estimated without ever building the Hessian matrix itself.
- **Power iteration** — a standard numerical method that finds the largest eigenvalue of a matrix by repeatedly applying the matrix to a vector and rescaling it.
- **Secant pair** — a pair consisting of a parameter step and the resulting change in gradient, used to estimate curvature along that step's direction.
- $\theta(t)$ — the model's parameter vector at training step t.
- $H(t)$ — the Hessian matrix of the loss function at $\theta(t)$.
- $S(t)$ — the sharpness (largest eigenvalue of $H(t)$) at step t.
- $m(t)$ — the stability margin, $2/\eta - S(t)$.
- $v_m(t)$ — the rate of change of the stability margin.
- $T_{\text{instab}}(t)$ — the estimated number of steps remaining before the margin reaches zero.

8 Conclusion

Large-scale AI training becomes unstable when the sharpness of its loss surface rises too close to a known threshold set by the learning rate. This is already understood in the research literature, but existing ways of watching for it live in software and either add extra computation or only look back after the fact. We have shown that the same early-warning idea used throughout the Arnold's Bifurcation Principle framework, applied here to a margin that shrinks as training sharpens rather than a curvature that flattens, can be tracked using only the gradient vectors a training run is already producing, through Hessian-free power iteration restricted to a small, well-chosen part of the model. A small, domain-blind hardware controller using this method could give real, real-time advance warning of an approaching training failure, in time to act before compute time and money are wasted on a run that is already past saving.

Acknowledgements

The author wishes to express his deepest gratitude to *Science Daily* magazine, an invaluable window into the global scientific landscape that continually introduces cutting-edge research across diverse fields and sparks the cross-disciplinary connections central to this work.

Most profoundly, this paper is dedicated to the more than 8,000 children the author has had the privilege of teaching over a 37-year career as a science instructor in Chennai, India. Their boundless curiosity, sharp questions, and vibrant energy provided the ultimate inspiration to keep the classroom alive. The daily joy of distilling complex, state-of-the-art scientific discoveries into intuitive, accessible ideas for young minds was the true crucible for this research framework. This work belongs, in spirit, to all of them.

Finally, the author acknowledges the invaluable assistance of advanced artificial intelligence models. While the underlying scientific concepts, theoretical syntheses, and original ideas presented herein are entirely and uniquely the author's own, these computational tools served as essential collaborators—helping to meticulously organize, refine, and assemble this manuscript brick by brick into its final technical structure.

References

[1] J. Cohen, S. Kaur, Y. Li, J. Z. Kolter, and A. Talwalkar, "Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability," International Conference on Learning Representations (2021).

[2] P. Marion, "Edge Flow: A Tractable and Predictive Continuous-Time Model for Gradient Descent at the Edge of Stability," arXiv:2606.18080 (2026).

[3] B. Ghorbani, S. Krishnan, and Y. Xiao, "An Investigation into Neural Net Optimization via Hessian Eigenvalue Density," International Conference on Machine Learning (2019).

[4] J. M. Cohen, B. Ghorbani, S. Krishnan, N. Agarwal, S. Medapati, M. Badura, D. Suo, D. Cardoze, Z. Nado, G. E. Dahl, J. Gilmer, "Adaptive Gradient Methods at the Edge of Stability," arXiv:2207.14484 (2022).

[5] B. A. Pearlmutter, "Fast Exact Multiplication by the Hessian," Neural Computation 6(1), 147–160 (1994).

[6] "MSign: An Optimizer Preventing Training Instability in Large Language Models via Stable Rank Restoration," arXiv:2602.01734 (2026).

[7] M. Wortsman, P. J. Liu, L. Xiao, K. Everett, A. Alemi, B. Adlam, J. D. Co-Reyes, I. Gur, A. Kumar, R. Novak, et al., "Small-scale proxies for large-scale Transformer training instabilities," arXiv:2309.14322 (2023).

[8] "Output Embedding Centering for Stable LLM Pretraining," arXiv:2601.02031 (2026).