We thank the reviewers for their critical feedback. Quite honestly there were large gaps in the explanation and analysis (as loosely clustered below: 10 of them), and we've made serious improvements to address nearly all:

1. Missing actionable use cases (R2, R3); implications of "50.7% helped" unclear (R3). Our paper aims to increase the understanding of neural networks in the same vein as insightful but not directly actionable papers like "Intriguing properties of Neural Networks" (Christian Szegedy et al, 2013) and "Opening the Black Box..." (Shwartz-Ziv and Tishby, 2017). We did find one use case (identify layers for freezing), but the primary goal of our paper is to provide a new measurement device (LC) and to clarify and update our mental models of NN training by using it.

2. Why is LC a good metric? Compare with FIM from [1,13] (R1). We believe both are useful. LC answers the specific question of "how do we allocate change in loss?" and is beneficial vs FIM because it is *grounded to the loss* and *signed*. *Grounded*: While [1] and [13] are illuminating, the FIM metric used may be rescaled arbitrarily (e.g. multiply one relu layer weights by 2 and next by 0.5: loss stays the same but FIM of each layer changes and total FIM changes). In contrast, LC is grounded, so its units are the same as loss (e.g. bits or nats for cross entropy) and its scale is fixed. *Signed*: FIM is unsigned and thus could not yield our "50.7% help" nor "some layers move backward" conclusion. We have clarified these points in the paper. *[See Update A in the figure below]*

3. Justify use of batch gradient (R1). This part was poorly presented; we have updated the text *[Update B]*. To measure training instead of training confounded with issues of memorization vs. generalization, we use the *train* gradients instead of *val*. Observations like the last layer hurting are also more surprising on train vs. val. We use *full-batch* gradients for analysis instead of single mini-batch gradient to measure learning in as noise-free a way as possible. Note that the optimizer only ever sees mini-batch gradients, as is the usual case for SGD or Adam.

4. Missing higher order terms, ignores curvature (R2); characterize approximation errors from first order (R1). This part was confusingly presented; text updated to clarify *[Update C]*. Curvature is not ignored, as LC is approximated with RK4, which is fourth-order accurate. First order was only mentioned as an example to illustrate the concept. Approximation errors on all networks with both first order and RK4 have been added *[Update D]* (good idea!). In short, first order can be off by a lot (it always overestimates the decrease in loss), but RK4 is within 0.3% total loss change.

5. "Help" and "hurt" are too simplistic and myopic, loss may locally increase then decrease later (R2). Each LC measure concerns the impact on loss at each specific iteration, and we agree that periods of hurting may be followed by helping and vice versa. We see this often (e.g. see Fig 3d). By summing LC over time, we can take arbitrarily less mypoic views (e.g. see Figs 4 and 5). We believe both microscopic and aggregate views can be useful.

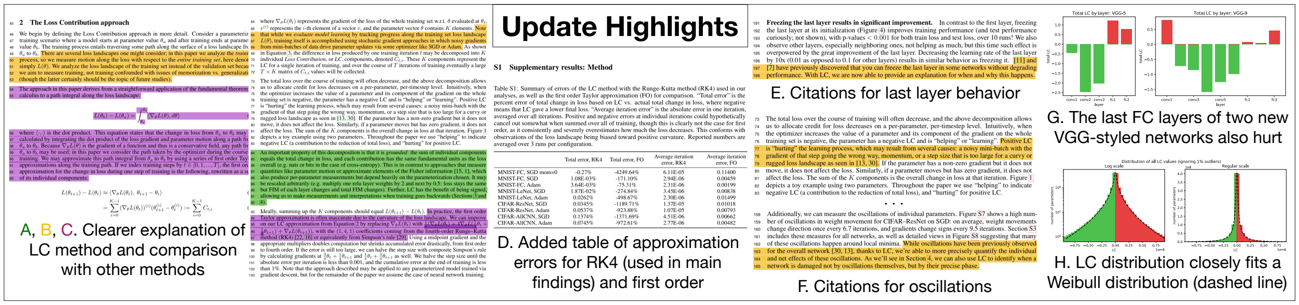
6. Last layer: different behavior and benefit of freezing already known (R2, R3). Thanks for these great references; we have added a discussion *[Update E]*. We build on this previous work by proposing a theoretical explanation (phase-delay) and validating the theory with careful experiments. See also: new multi-layer results in #7.

7. How universally does last layer hurt? (R1). We don't claim this to be a universal phenomenon, but when it occurs LC allows us to uncover it and take beneficial actions such as freezing layers. It is also not just a one-off observation: in addition to ResNet, it also happens for AllCNN (mentioned in paper) where the last layer is conv, and we have now measured it in VGG-based models as well, where the last 2-3 FC layers show the phenomenon *[Update G]*.

8. Citations, oscillations not novel (R3). Thanks for these great references; we had not meant to claim discovery of oscillation and have updated the text to clarify *[Update F]*. LC merely tracks such behavior on a per-parameter level and can expose, for example, whether a parameter or layer is slowly making progress or the reverse over time.

9. Intuition of results of freezing the first layer (R3). Indeed, given the observed behavior of freezing the first layer (that other layers then help less), we should be careful not to over-interpret a positive per-layer LC as directly indicating that that layer should be removed or frozen. The interaction effects between layers would seem to matter a lot, though we do not fully understand these effects yet (though our phase-delay results are a step in the right direction).

10. Learning is heavy tailed section not precise (R3). Thanks for the suggestion. We've updated it to a more precise characterization: the LC distribution is closely modeled by a heavy-tailed Weibull distribution *[Update H, dashed line shows the Weibull distribution, parameter $k = 0.53$ in example]*. This refines a detail in our mental model of training.



2 The Loss Contribution approach

We begin by defining the Loss Contribution approach in more detail. Consider a parameteriz training scenario where a model starts at parameter value θ_a and after training ends at parame value θ_b. The training process entails traversing some path along the surface of a loss landscape frc θ_a to θ_b. There are several loss landscapes one might consider; in this paper we analyze the *traini* process, so we measure motion along the loss with respect to the *entire training set*, here denot simply $L(\theta)$. We analyze the loss landscape of the training set instead of the validation set becau we aim to measure training, not training confounded with issues of memorization vs. generalizati (though the latter certainly should be the topic of future studies).

The approach in this paper derives from a straightforward application of the fundamental theorem calculus to a path integral along the loss landscape:

$$L(\theta_b) - L(\theta_a) = \int_{\theta_a}^{\theta_b} \langle \nabla_\theta L(\theta), d\theta \rangle$$

where $\langle \cdot, \cdot \rangle$ is the dot product. This equation states that the change in loss from θ_a to θ_b may calculated by integrating the dot product of the loss gradient and parameter motion along a path frc θ_a to θ_b. Because $\nabla_\theta L(\theta)$ is the gradient of a function and thus is a conservative field, any path frc θ_a to θ_b may be used; in this paper we consider the path taken by the optimizer during the course training. We may approximate this path integral from θ_a to θ_b by using a series of first order Tayl approximations along the training path. If we index training steps by $t \in [0, 1, ..., T]$, the first orc approximation for the change in loss during one step of training is the following, rewritten as a su of its individual components:

$$\begin{aligned} L(\theta_{t+1}) - L(\theta_t) &\approx \langle \nabla_\theta L(\theta_t),\ \theta_{t+1} - \theta_t \rangle \\ &= \sum_{i=0}^{K-1} (\nabla_\theta L(\theta_t))^{(i)} (\theta_{t+1}^{(i)} - \theta_t^{(i)}) := \sum_{i=0}^{K-1} C_{t,i} \end{aligned}$$

where $\nabla_\theta L(\theta_t)$ represents the gradient of the loss of the whole training set w.r.t. θ evaluated at θ_t, $v^{(i)}$ represents the i-th element of a vector v, and the parameter vector θ contains K elements. Note that while we *evaluate model learning* by tracking progress along the training set loss landscape $L(\theta)$, training itself is accomplished using stochastic gradient approaches in which noisy gradients from mini-batches of data drive parameter updates via some optimizer like SGD or Adam. As shown in Equation 3, the difference in loss produced by one training iteration t may be decomposed into K individual *Loss Contribution*, or *LC*, components, denoted $C_{t,i}$. These K components represent the LC for a single iteration of training, and over the course of T iterations of training eventually a large $T \times K$ matrix of $C_{t,i}$ values will be collected.

The total loss over the course of training will often decrease, and the above decomposition allows us to allocate credit for loss decreases on a per-parameter, per-timestep level. Intuitively, when the optimizer increases the value of a parameter and its component of the gradient on the whole training set is negative, the parameter has a negative LC and is "helping" or "learning". Positive LC is "hurting" the learning process, which may result from several causes: a noisy mini-batch with the gradient of that step going the wrong way, momentum, or a step size that is too large for a curvy or rugged loss landscape as seen in [13, 30]. If the parameter has a non-zero gradient but it does not move, it does not affect the loss. Similarly, if a parameter moves but has zero gradient, it does not affect the loss. The sum of the K components is the overall change in loss at that iteration. Figure 1 depicts a toy example using two parameters. Throughout the paper we use "helping" to indicate negative LC (a contribution to the reduction of total loss), and "hurting" for positive LC.

An important property of this decomposition is that it is *grounded*: the sum of individual components equals the total change in loss, and each contribution has the same fundamental units as the loss overall (e.g. nats or bits in the case of cross-entropy). This is in contrast to approaches that measure quantities like parameter motion or approximate elements of the Fisher information [15, 1], which also produce per-parameter measurements but depend heavily on the parameterization chosen. It may be rescaled arbitrarily (e.g. multiply one relu layer weights by 2 and next by 0.5: loss stays the same but FIM of each layer changes and total FIM changes). Further, LC has the benefit of being signed, allowing us to make measurements and interpretations when training goes backwards (Sections 3 and 4).

Ideally, summing up the K components should equal $L(\theta_{t+1}) - L(\theta_t)$. In practice, the first order Taylor approximation is often inaccurate due to the curvature of the loss landscape. We can improve on our LC approximation from Equation 2 by replacing $\nabla_\theta L(\theta_t)$ with $\frac{1}{6}(\nabla_\theta L(\theta_t) + 4\nabla_\theta L(\frac{1}{2}\theta_t + \frac{1}{2}\theta_{t+1}) + \nabla_\theta L(\theta_{t+1}))$, with the $(1, 4, 1)$ coefficients coming from the fourth-order Runge–Kutta method (RK4) [22, 16] or equivalently from Simpson's rule [29]. Using a midpoint gradient and the appropriate multipliers doubles computation but shrinks accumulated error drastically, from first order to fourth order. If the error is still too large, we can halve the step size with composite Simpson's rule by calculating gradients at $\frac{3}{4}\theta_t + \frac{1}{4}\theta_{t+1}$ and $\frac{1}{4}\theta_t + \frac{3}{4}\theta_{t+1}$ as well. We halve the step size until the absolute error per iteration is less than 0.001, and the cumulative error at the end of training is less than 1%. Note that the approach described may be applied to any parameterized model trained via gradient descent, but for the remainder of the paper we assume the case of neural network training.

A, B, C. Clearer explanation of LC method and comparison with other methods

Update Highlights

S1 Supplementary results: Method

Table S1: Summary of errors of the LC method with the Runge-Kutta method (RK4) used in our analyses, as well as the first order Taylor approximation (FO) for comparison. "Total error" is the percent error of total change in loss based on LC vs. actual total change in loss, where negative means that LC gave a lower final loss. "Average iteration error" is the absolute error in one iteration, averaged over all iterations. Positive and negative errors at individual iterations could hypothetically cancel out somewhat when summed over all of training, though this is clearly not the case for first order, as it consistently and severely overestimates how much the loss decreases. This conforms with observations of the loss landscape being biased toward positive curvature. Reported numbers are averaged over 3 runs per configuration.

	Total error, RK4	Total error, FO	Average iteration error, RK4	Average iteration error, FO
MNIST-FC, SGD mom=0	-0.27%	-4249.64%	6.11E-05	0.11400
MNIST-FC, SGD	1.08E-03%	-171.10%	2.94E-06	0.00459
MNIST-FC, Adam	3.64E-03%	-75.31%	2.31E-06	0.00199
MNIST-LeNet, SGD	1.87E-02%	-274.84%	3.45E-06	0.00838
MNIST-LeNet, Adam	0.0262%	-498.67%	2.30E-06	0.01499
CIFAR-ResNet, SGD	0.0345%	-1189.71%	1.37E-05	0.01018
CIFAR-ResNet, Adam	0.0537%	-923.88%	1.07E-05	0.00793
CIFAR-AllCNN, SGD	0.1374%	-1371.69%	4.51E-06	0.00662
CIFAR-AllCNN, Adam	0.0745%	-972.61%	2.77E-06	0.00482

D. Added table of approximation errors for RK4 (used in main findings) and first order

Freezing the last layer results in significant improvement. In contrast to the first layer, freezing the last layer at its initialization (Figure 4) improves training performance (and test performance curiously; not shown), with p-values < 0.001 for both train loss and test loss, over 10 runs! We also observe other layers, especially neighboring ones, not helping as much, but this time such effect is overpowered by the great improvement of the last layer. Decreasing the learning rate of the last layer by 10x (0.01 as opposed to 0.1 for other layers) results in similar behavior as freezing it. [11] and [7] have previously discovered that you can freeze the last layer in some networks without degrading performance. With LC, we are now able to provide an explanation for when and why this happens.

E. Citations for last layer behavior

The total loss over the course of training will often decrease, and the above decomposition allows us to allocate credit for loss decreases on a per-parameter, per-timestep level. Intuitively, when the optimizer increases the value of a parameter and its component of the gradient on the whole training set is negative, the parameter has a negative LC and is "helping" or "learning". Positive LC is "hurting" the learning process, which may result from several causes: a noisy mini-batch with the gradient of that step going the wrong way, momentum, or a step size that is too large for a curvy or rugged loss landscape as seen in [13, 30]. If the parameter has a non-zero gradient but it does not move, it does not affect the loss. Similarly, if a parameter moves but has zero gradient, it does not affect the loss. The sum of the K components is the overall change in loss at that iteration. Figure 1 depicts a toy example using two parameters. Throughout the paper we use "helping" to indicate negative LC (a contribution to the reduction of total loss), and "hurting" for positive LC.

. . .

Additionally, we can measure the oscillations of individual parameters. Figure S7 shows a high number of oscillations in weight movement for CIFAR–ResNet on SGD: on average, weight movements change direction once every 6.7 iterations, and gradients change signs every 9.5 iterations. Section S3 includes these measures for all networks, as well as detailed views in Figure S8 suggesting that many of these oscillations happen around local minima. While oscillations have been previously observed for the overall network [30, 13], thanks to LC, we're able to more precisely quantify the individual and net effects of these oscillations. As we'll see in Section 4, we can also use LC to identify when a network is damaged not by oscillations themselves, but by their precise phase.

F. Citations for oscillations

G. The last FC layers of two new VGG-styled networks also hurt

H. LC distribution closely fits a Weibull distribution (dashed line)