Temperature Balancing, Layer-wise Weight Analysis, and Neural Network Training

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Regularization in modern machine learning is crucial, and it can take various forms in algorithmic design: training set, model family, error function, regularization terms, and optimizations. The learning rate, which can be interpreted as a *temperature* parameter within the statistical mechanics of learning, plays a crucial role in training deep neural networks. Indeed, many widely adopted strategies define the decay of the learning rate over time, using either a global learning rate or one that varies for each parameter, which can be interpreted as decreasing the temperature. This paper proposes a middle-ground approach for temperature balancing called `TempBalance`. It is based on the theory of heavy-tail self-regularization (HT-SR), and it is a simple yet effective layer-wise policy applicable to general global temperature assignments in deep learning regularization. Our main contributions are as follows: (i) In addition to following a learning rate schedule, we suggest balancing the learning rate across each layer, an approach that has received less attention compared to global or parameter-wise learning rate allocation. (ii) We demonstrate that HT-SR-motivated capacity control metrics characterize the layers to achieve maximal temperature balance during model training, resulting in improved performance during testing. We implement `TempBalance` on CIFAR10, CIFAR100, SVHN, and TinyImageNet datasets using ResNets, VGGs and WideResNets with various depths and widths. Our results show that `TempBalance` significantly outperforms ordinary `SGD` and carefully-tuned spectral norm regularization, which is a closely related regularization technique. We also show that `TempBalance` outperforms a number of state-of-the-art optimizers and learning rate schedulers.

1 Introduction

Having a learning rate schedule that gradually decreases over time is crucial for the convergence and performance of state-of-the-art machine learning algorithms. Indeed, many optimization algorithms essentially boil down to designing the progression of parameter updates, as realized by different learning rate schedules [1–4]. Common schedules assign a global model learning rate per epoch, where the same learning rate is used for all layers in the model. This includes the family of cyclical learning rates [3], and parameter-wise learning rate schedules like Adam [2] and its variants [5, 6]. However, such a global learning rate schedule does not take into account the structural characteristics of neural networks. At the same time, parameter-wise learning rate schedules have long been conjectured to have worse generalization performance than carefully tuned `SGD` optimizers [7], and storing both first and second-order moments for each parameter can lead to *significantly* increased memory consumption [8]. As mentioned in Smith et al. [9], storing the whole Megatron-Turing NLG requires 10 terabytes of aggregate memory, and the `Adam` optimizer's first and second-order moments [2] consume 40% of it. Nonetheless, improving parameter-wise learning rate schedules is an active field of study [4, 5, 10, 11].

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

A largely under-explored idea is to assign layer-wise learning rates to reconcile the two extremes of setting a single global learning rate or assigning fine-grained parameter-level learning rates. This learning rate assignment method does not require much storage cost and can assign different training speeds to different layers. However, existing layer-wise schemes are often introduced as an additional part of hyperparameter sweeping, and most lack a strong theoretical foundation. For instance, layer-wise learning rates can increase test accuracy in transfer learning [12] and domain adaptation [13], but these learning rates are often empirically tuned. More recently, Chen et al. [14], motivated by the idea that lower-level layers are domain-specific and higher-level layers are task-specific, automates the search for the optimal set of learning rates. However, the authors find the nested, bi-level optimization scheme to be too computationally expensive in practice [15]. `AutoLR` also automatically tunes its layer-wise learning rates according to the "role" of each layer [16]. The method is validated almost entirely by empirical results, further explained by layer-wise weight variations. While the authors attempt to assign a different starting learning rate to each layer, the learning rate for each layer continues to stay largely constant throughout training. `LARS` [17, 18] is another method to assign layer-wise learning rate. It is based on the "trust ratio", defined as the ratio of weight norm to gradient update norm of each layer, and it is specifically used in large batch training to avoid the gradient diverge.

In this paper, we propose `TempBalance`, a simple yet effective layer-wise learning rate assignment regularization method. `TempBalance` adopts a *statistical physics viewpoint* of learning and optimization [19–22], and it views the learning rate as a "temperature parameter", which refers to some quantity related to the empirical noise/stochasticity introduced in the learning process. From this viewpoint, what matters in `SGD` training is the noise scale (which is the same as the noise scale mentioned in Smith and Le [23], Smith et al. [24] that can be written as a function of learning rate, batch size and momentum), instead of the learning rate *per se*. Further, a series of recent papers [25–27] point out that neural network training can be viewed as a balance between *temperature-like parameters* and *load-like parameters*, where a load-like parameter refers to some quantity related to the quantity/quality of the data, relative to the size of the model. For instance, Yang et al. [26] vary load and temperature parameters to provide a comprehensive taxonomy of NN loss landscapes, showing sharp phase transitions between different types of loss landscapes. In this paper, building on this line of research on temperature parameters, we further use ideas and measurements from Heavy-Tail Self Regularization (HT-SR) Theory [25, 28–31] to characterize the quality of each layer and then assign *layer-wise temperature* (i.e., layer-wise learning rate) based on their heavy-tail (HT) characterizations. We discuss the significance of HT-SR and its connection to layer-wise temperature in the following paragraph.

HT-SR theory. HT-SR theory suggests that as layer weight matrices train for a longer period, they start to show strong correlations, resulting in the HT structure of the Empirical Spectrum Density (ESD) for each layer. To obtain this ESD, we take a neural network with L layers and its corresponding weight matrices $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_L$ with shape $n \times m$ (where $n \geq m$). For the i-th layer, we calculate the eigenvalues of its correlation matrix $\mathbf{X}_i = \mathbf{W}_i^T \mathbf{W}_i$ and then plot the ESD for that layer, which, upon training, will gradually change to have an HT structure by HT-SR theory [28]. We can then fit a power-law (PL) distribution to the HT part of the ESD, and extract its exponent as, namely, `PL_Alpha`. The fitted PL will have the following formula:

$$p(\lambda) \propto \lambda^{-\alpha}, \quad \lambda_{min} < \lambda < \lambda_{max}. \tag{1}$$

The `PL_Alpha` metric measures the PL exponent of the weight matrices' ESD, and its underlying motivation stems from random matrix theory and statistical physics [29, 30].

The `PL_Alpha` metric has been shown to predict the trends in the test accuracy of state-of-the-art computer vision (CV) and natural language processing (NLP) neural networks, without even the need for access to training or testing data [30, 32]. According to Martin et al. [30], one can aggregate `PL_Alpha`'s for different layers either by simple averaging or weighted averaging, and they can all predict test accuracy in different cases [30, 32]. Furthermore, the *layer-wise* nature of `PL_Alpha` makes it a fine-grained metric that can be used to assess the quality of individual layers of the network. Thus, in this paper, we extend HT-SR to training, and we exploit the layer-wise information provided by `PL_Alpha` to determine the layer-wise learning rates for better test accuracy.

Note that `PL_Alpha` is not the only way to measure the HT structure. Several recent papers [33–35] use different HT metrics to measure the spectral of several "important matrices" (such as input/output covariance matrices, Fisher Information Matrices and Hessian), and we show in Appendix A that

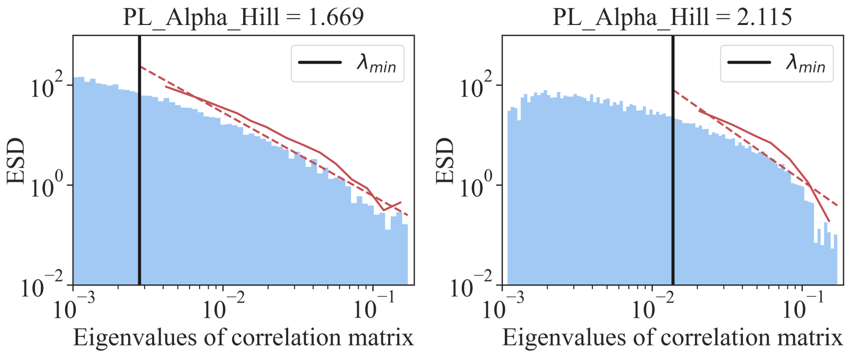


Figure 1: Examples of power-law (PL) fitting using the Hill estimator. Blue histograms depict the empirical spectral densities (ESDs). Vertical black lines indicate the lower threshold λ_{min} used to truncate the full ESDs and extract the tail portion. Solid red curves represent the tail part of the ESDs truncated by λ_{min}, while dashed red curves represent the fitted heavy-tailed (HT) distributions. The left shows a more HT ESD, requiring a relatively lower learning rate. The right one shows less HT ESD, which requires a relatively higher learning rate. Unlike prior work, we do not aim to find the "optimal" PL exponent. Instead, we use the PL exponent to *rank* ESDs to find layers that need higher/lower learning rates. These two ESDs correspond to two layers of a ResNet18 model trained on TinyImageNet.

these HT phenomena, measured in different ways on different matrices, are closely related to each other. On the other hand, this also means that the "absolute value" of `PL_Alpha` is unimportant, as optimal PL exponents estimated by different algorithms can be different [30, 33]. It turns out that what matters the most, as we show in this paper, is the layer-wise quality *ranked* by the PL exponent: layers with a smaller `PL_Alpha` to be relatively more "overtrained" and those with a larger `PL_Alpha` to be relatively more "undertrained". This observation leads to a simple and efficient way to *balance* layer-wise learning rates: assigning a lower learning rate to overtrained layers and a larger learning rate to undertrained layers using `PL_Alpha` (see Figure 1). On top of this, we can grid-search the *global* learning rate on validation data, which is standard practice and is more efficient than grid-searching the layer-wise learning rates. We use this combination of assigning layer-wise learning rates using `PL_Alpha` and grid-searching the base learning rate to avoid deciding the "optimal PL exponent", which can be tricky due to different ways of measuring HT. Indeed, there are different ways to measure `PL_Alpha` [28], and we use the Hill estimator [36], which shows stable performance in our experiments. Thus, we call our version of `PL_Alpha` the `PL_Alpha_Hill` metric, and we use that for the remaining paper. Further, we use a *scale-free* way to map the estimated `PL_Alpha_Hill` to the learning rate, meaning that arbitrary linear scaling on the estimated `PL_Alpha_Hill` (either due to the choice of the estimator or noisy measurements) does not affect the assigned learning rates.

Another common way to change the ESD of weights is to constrain the spectral norm (i.e., the largest eigenvalue) using spectral norm regularization (`SNR`) [37, 38]. `SNR` provides a different form of regularization compared to HT-SR because it regulates the largest eigenvalue instead of the ESD slope (i.e., the `PL_Alpha_Hill` metric). It has been demonstrated that spectral norm and `PL_Alpha_Hill` serve distinct roles in generalization, and their combined form yields optimal predictions for test accuracy trends [28, 30–32]. Our paper, on the other hand, demonstrates that `TempBalance` outperforms `SNR` in training deep neural networks in most cases. Moreover, when these two regularization methods are combined during training, they result in optimal test accuracy, thereby confirming their complementary roles. As mentioned by Martin and Mahoney [31], Yang et al. [32], spectral norm and `PL_Alpha_Hill` measure the scale and the shape of a ESD respectively, and regulating both the scale and shape is crucial for achieving better ESD regularization. We also provide ablation studies on several layer-wise metrics for assigning layer-wise learning rates, including spectral norm, and we show that `PL_Alpha_Hill` performs the best among them.

Contributions. The following summarizes our main contributions:

- We propose a simple yet effective layer-wise learning rate schedule `TempBalance` based on HT-SR theory. We empirically found two insights. First, the mapping from `PL_Alpha_Hill` to learning rates should be scale-free, meaning that arbitrary linear scaling on the estimated PL exponent should not change the learning rate assignment. Second, searching for the minimum eigenvalue $\lambda_{\min}$, a standard practice in PL fitting [28, 39, 40], leads to unstable training. We instead fix $\lambda_{\min}$ as the medium of the ESD.

- We compare `TempBalance` to ordinary stochastic gradient descent (`SGD`) and `SNR` on various training tasks. This includes (1) different network architectures, such as ResNet, VGG, WideResNet, (2) different datasets, such as CIFAR10, CIFAR100, SVHN, TinyImageNet, and (3) ablation studies, such as varying widths, depths, initial learning rates and HT-SR layer-wise metrics. Compared to ordinary `SGD`, `TempBalance` achieves higher test accuracy by setting layer-wise learning rates. Compared to `SNR`, `TempBalance` performs better by providing a more fine-grained regularization on *shape/slope* instead of norm. We also show that combining `TempBalance` and `SNR` leads to further improved accuracy, verifying their complementary roles in informing deep learning training.
- We compare `TempBalance` to a range of state-of-the-art optimizers and learning rate schedulers, including `SGDR` [10], `SGDP` [41], `Lookahead` [42] and `LARS` [17, 18] on ResNet18 and ResNet34 trained on CIFAR100. We show that `TempBalance` achieves the highest test accuracy. We do careful hyperparameter tuning for all baselines. All results are obtained from five random seeds.
- We use ablation studies to show that `PL_Alpha_Hill` provides the best test accuracy among a few layer-wise metrics considered in HT-SR [30, 32]. We also show that `TempBalance` maintains stable performance over `SGD` baselines when the model size changes. Furthermore, we show visualization results in Appendix B, verifying that `TempBalance` controls ESDs during training.

2 Related Work

Here we give an overview of the statistical mechanics of learning and recent progress in theoretical and empirical studies on generalization metrics and their applications.

2.1 Statistical mechanics of learning and HT-SR

Our paper is motivated by statistical mechanics of learning [43–45], and especially by works that connect load-like [43, 46, 47] and temperature-like parameters [19, 48] to neural networks. According to prior works in this area [25, 26], a temperature-like parameter represents the amount of noise and disturbance in the iteration of `SGD`, such as learning rate, weight decay parameter, and batch size. A load-like parameter represents the quantity and/or quality of data relative to the size of the learning model. To measure the quality of trained neural networks, Martin and Mahoney [28] introduce HT-SR theory, showing that the weight matrices of deep neural networks exhibit heavy-tailed empirical spectral densities. In subsequent papers, HT-SR has been applied to predicting trends in test accuracy of large-scale neural networks in both CV and NLP [30–32], but it has yet to be systematically incorporated to novel training algorithms. Recently, more and more papers realize the important connections between deep neural networks and statistical mechanics of learning. To name a few, Yang et al. [26] use load and temperature parameters to study a wide range of loss landscapes, providing a taxonomy from the perspective of *global structure* of a loss landscape. On the theory side, Baity-Jesi et al. [49] investigates the glassy behavior of neural networks, and Barbier et al. [50] derives the optimal generalization error of generalized linear systems. More recently, Sorscher et al. [51] studies easy versus hard samples used in training and design a "data-pruning" method. Zhou et al. [27] establishes a "three-regime model" in network pruning, unifying multiple practical hyperparameter tuning methods in a principled way.

2.2 Generalization measures

Note that the search for effective and robust generalization metrics has been the focus of several recent theoretical and empirical works [26, 30, 32, 52–54]. Several recent papers apply metric-informed training and architecture search, such as those based on Hessian [4, 55–57], spectral norm [37, 38], stable rank [58] and the spectrum of neural tangent kernel [59]. However, the most empirically successful generalization metrics, such as those based on the PAC-Bayes bounds [60–63], do not straightforwardly transfer to layer-wise quality metrics because such generalization metrics often study the whole neural network as an architecture-free function and lack the fine granularity to unveil the quality of each layer. Also, it has been mentioned in the literature [52] that (1) directly regularizing

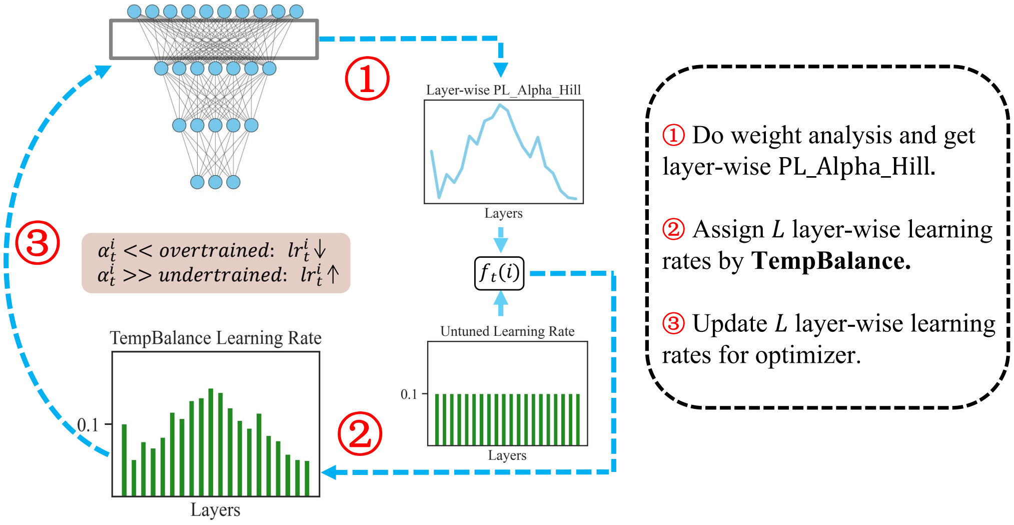

①
Layer-wise PL_Alpha_Hill
Layers
$f_t(i)$
Untuned Learning Rate
0.1
Layers
③
α_t^i << overtrained: lr_t^i ↓
α_t^i >> undertrained: lr_t^i ↑
TempBalance Learning Rate
0.1
Layers
②
① Do weight analysis and get layer-wise PL_Alpha_Hill.
② Assign L layer-wise learning rates by TempBalance.
③ Update L layer-wise learning rates for optimizer.

Figure 2: The pipeline diagram of `TempBalance`. For each epoch, `TempBalance` completes 3 steps: (i) Do weight analysis for all layers and get layer-wise `PL_Alpha_Hill`. (ii) Leverage layer-wise `PL_Alpha_Hill` to assign learning rate for each layer while maintaining their change range expectation roughly equal to the baseline across layers. (iii) Update the optimizer for the next epoch.

Algorithm 1: `TempBalance`

```
Input: $M$: Deep Neural Network,   $T$: Total training epoch,   $t$: Current epoch,
       $\alpha_t^i$: $i_{th}$ layer's PL_Alpha_Hill at epoch $t$,   $\eta_t$: Baseline global learning rate at epoch $t$,
       $s_1, s_2$: Minimum and maximum scaling ratio,   $f_t$: Learning rate schedule function
Initialize model $M$;
for $t \leftarrow 0$ to $T$ do
    Compute $\alpha_t^i$ for all layers using the Hill estimator;
    Leverage all $\alpha_t^i$ and adopt $f_t$ in (2) to assign per-layer learning rate $f_t(i)$ between $s_1\eta_t$ and
      $s_2\eta_t$ for the next epoch;
    Update the optimizer for the next epoch;
end
```

generalization metrics can lead to difficulty in training, (2) evaluating these regularization methods may be hard due to the existence of *implicit regularization* in `SGD`, and (3) these metrics, especially norm-based metrics, cannot be expected to correlate with test accuracy causally [53], making the link between these generalization metrics and practical training methods nuanced. It will be clear in the next section that we do not regularize ESD metrics directly. Instead, we change learning rates to modify ESDs.

3 The `TempBalance` Algorithm

In this section, we introduce our simple yet effective method `TempBalance`, based on the generalization metric `PL_Alpha_Hill` from HT-SR theory. For a neural network, different layers tend to have different `PL_Alpha_Hill`, [25, 28]: a layer with larger `PL_Alpha_Hill` indicates it is relatively undertrained while a smaller `PL_Alpha_Hill` means it is relatively overtrained. A natural idea is to adjust the degree of learning among different layers to get a balance: for a layer whose `PL_Alpha_Hill` is too large, we could assign a larger learning rate to accelerate its learning, and vice versa. We keep the expectation of the learning rate change range across all layers roughly equal to a baseline *global* learning rate, making it easier to tune the global learning rate and compare it with other baseline methods. The intuition of our method is transferring one layer's learning rate to another and hence, `TempBalance`. The pipeline is in Figure 2.

We provide the details of `TempBalance` in Algorithm 1. Based on `PL_Alpha_Hill` in different layers, we use the learning rate schedule function f_t to map the i_{th} layer to a particular learning rate $f_t(i)$ in epoch t. We adopt f_t as a linear map between the layer-wise `PL_Alpha_Hill` and the final layer-wise learning rate, which has the following formula:

$$f_t(i) = \eta_t \cdot \left[\frac{\alpha_t^i - \alpha_t^{min}}{\alpha_t^{max} - \alpha_t^{min}} (s_2 - s_1) + s_1 \right], \tag{2}$$

where η_t means the base global learning rate in epoch t, (s_1, s_2) are the minimum and maximum learning rate scaling ratio relative to η_t, α_t^i represents the layer i's `PL_Alpha_Hill` at the beginning of epoch t, and $(\alpha_t^{min}, \alpha_t^{max})$ denote the minimum and maximum `PL_Alpha_Hill` across all the layers in epoch t. Using (2), we ensure that the new learning rate $f_t(i)$ is a scaled version of the original base learning rate η_t and is always inside the interval $[s_1\eta_t, s_2\eta_t]$. We only consider (s_1, s_2) such that $\frac{s_1+s_2}{2} = 1$, e.g., (0.5, 1.5) or (0.8, 1.2).

To fit the power law distribution $p(\lambda)$ defined in (1), we use the famous Hill estimator [36] [64]. For the i-th layer, suppose the weight matrix is $\mathbf{W}_i$ and the correlation matrix $\mathbf{W}_i^\top \mathbf{W}_i$ has ascending eigenvalues $\{\lambda_i\}_{i=1}^n$. Then, the Hill estimator calculates `PL_Alpha_Hill` using the following:

$$\texttt{PL\_Alpha\_Hill} = 1 + \frac{k}{(\sum_{i=1}^{k} ln \frac{\lambda_{n-i+1}}{\lambda_{n-k}})}, \tag{3}$$

where k is the adjustable parameter, and we adopt $k = \frac{n}{2}$ in our experiments. Note that changing k essentially changes the lower eigenvalue threshold λ_{min} for (truncated) PL estimation, as shown by the vertical black line in Figure 1. Choosing $k = \frac{n}{2}$ means using the largest half of the eigenvalues to estimate the slope. We empirically find that fixing k for all layers leads to more stable performance than searching k for different layers (e.g., optimizing k using the Kolmogorov–Smirnov test [40].)

One advantage of mapping `PL_Alpha_Hill` to learning rates using (2) is that the scale of `PL_Alpha_Hill` is unimportant, i.e., linearly scaling `PL_Alpha_Hill` arbitrarily does not change the learning rate assignment because the linear scaling cancels each other in (2). This can maximally reduce the artifact of estimating the ESD PL exponent/slope due to estimation noise, which has been found to be a tricky issue in practice [28, 31].

4 Empirical results

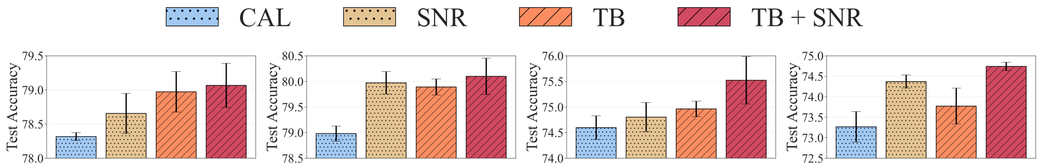
In this section, we first give full details of the experimental setup (Section 4.1) and compare our method `TempBalance` to a few baselines (Section 4.2). Then, in Section 4.3, we perform ablation studies on varied initial learning rates, model widths and HT-SR layer-wise metrics.

4.1 Experimental setup

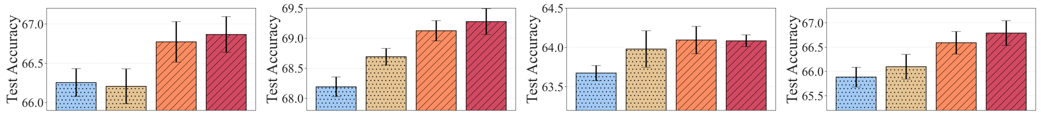
Datasets. We consider CIFAR100, CIFAR10, SVHN and Tiny ImageNet (TIN) [65–69]. CIFAR100 consists of 50000 pictures for training and 10000 pictures for testing with 100 categories. CIFAR10 consists of 50000 pictures for training and 10000 pictures for testing with 10 categories. SVHN consists of 73257 pictures for training and 26032 pictures for testing with 10 categories. Tiny ImageNet consists of 100000 pictures for training and 10000 images for testing with 200 classes.

Models. We mainly consider three types of deep neural networks, VGG, ResNet and WideResNet (WRN) [70–72]. For each network, we consider two different size options. For VGG, we consider VGG16 and VGG19. For ResNet, we consider ResNet18 and ResNet34. For WideResNet, we consider WRN16-8 and WRN28-6. Also, for ResNet and VGG, we consider three different widths for ablation studies.

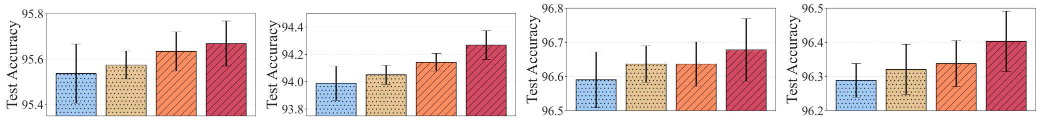
Hyperparameters. One baseline is ordinary `SGD` training with a cosine annealing learning rate schedule (`CAL`), which follows the formula: $\eta_t = \frac{\eta_0}{2}\left(1 + \cos\left(\frac{t \cdot \pi}{T}\right)\right)$, where t is the current epoch, T represents the total training epochs and η_0 is the initial learning rate. We grid-search the optimal initial (base) learning rate η_0 for the `CAL` baseline, using the grid $\{0.05, 0.1, 0.15\}$ for ResNet and $\{0.025, 0.05, 0.1\}$ for VGG. The momentum and weight decay are 0.9 and 5×10^{-4}, respectively, which are both standard choices.



(a) ResNet18, CIFAR100 (b) ResNet34, CIFAR100 (c) VGG16, CIFAR100 (d) VGG19, CIFAR100



(e) ResNet18, TIN (f) ResNet34, TIN (g) WRN16-8, TIN (h) WRN28-6, TIN



(i) ResNet18, CIFAR10 (j) VGG16, CIFAR10 (k) ResNet18, SVHN (l) VGG16, SVHN

Figure 3: **(Main result).** Comparing our method `TempBalance` (`TB`) to `CAL` and `SNR`. Our method `TempBalance` outperforms `CAL` and `SNR` in almost all the settings except for VGG19 and ResNet 34 on CIFAR 100. For all experiments, combining `TempBalance` and `SNR` (`TB+SNR`) yields the best performance. All baselines are carefully tuned. All results are obtained by running five random seeds. See Appendix C for the details in all hyperparameters.

Another baseline is called spectral norm regularization (`SNR`). Prior work uses the following SNR objective function [37]:

$$\min_{\Theta} \frac{1}{n} \sum_{i=1}^{n} l\left(f_{\Theta}\left(\boldsymbol{x}_i\right), \boldsymbol{y}_i\right) + \frac{\lambda_{sr}}{2} \sum_{l=1}^{L} \sigma\left(W_l\right)^2, \tag{4}$$

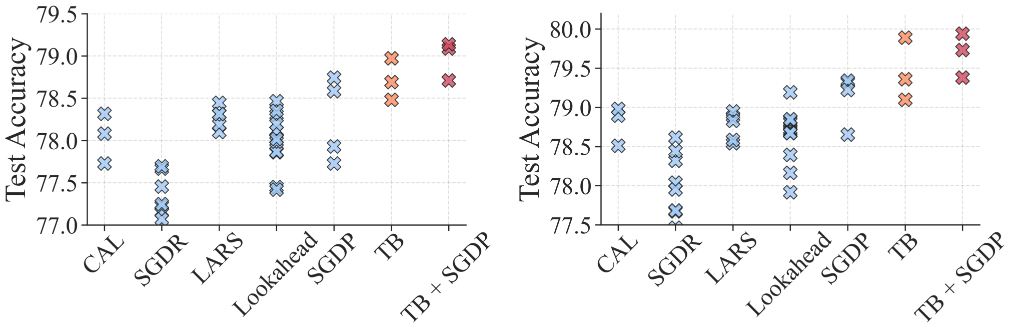
where λ_{sr} is the spectral norm regularization coefficient, $\sigma(W_l)$ is the largest eigenvalue, i.e, spectrum norm of weight matrix $\mathbf{W}_l$, and L is the number of layers. We use the power iteration method to calculate $\sigma(W_l)$ in our experiments. For `SNR`, we grid-search the optimal regularization coefficient λ_{sr}, and we again adopt the `CAL` schedule for `SNR`, similar to the `CAL` baseline.

To make our results fully reproducible, we report in Appendix C all hyperparameters, random seeds, and all numerical values of experimental results shown in the figures.

4.2 Comparing `TempBalance` and multiple baseline methods.

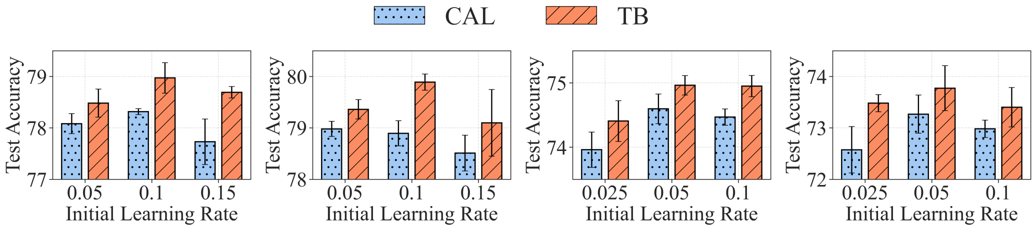
First, we compare `TempBalance` to two baseline training methods. See results in Figure 3. In the figure, `CAL` means `SGD` training with a `CAL` learning rate schedule, `SNR` means `SGD` trained with spectral norm regularization. `TB` means our method `TempBalance`, and `TB + SNR` means `TempBalance` combined with `SNR`. All error bars are obtained from five random seeds. From Figure 3, we see that `TempBalance` outperforms the `CAL` baseline in all settings. In almost all cases, it performs better than `SNR` baseline. When `TempBalance` does not outperform `SNR`, combining `SNR` with `TempBalance` leads to better test accuracy.

Second, we compare our method to a number of optimizers and learning rate schedulers that are not necessarily related to ESD of weights. These include `SGDR` [10], `SGDP` [41], `Lookahead` [42] and `LARS` [17, 18], and we compare these baselines with `TempBalance` for ResNet18 and ResNet34 trained on CIFAR100. `SGDR` is stochastic gradient descent with warm restarts. `SGDP` modifies the ordinary `SGD` to compensate for the effect of increasing weight norm. `Lookahead` [42] modifies `SGD` by letting each gradient update approximate the future trajectory of multiple updates. `LARS` assigns layer-wise learning rates based on the so-called “trust-ratio” and is the closest to our method. Results in Figure 4 show that `TempBalance` outperforms these baselines, and `TempBalance` combined with `SGDP` is the best-performing method. The crosses on each column represent training runs with different hyperparameters. Note that there are several other methods based on modifying the `Adam`



(a) ResNet18, CIFAR100 (b) ResNet34, CIFAR100

Figure 4: **(More baseline optimizers).** Comparing our method `TempBalance` (`TB`) to cosine annealing (`CAL`) baseline and other state-of-the-art optimizers and learning rate schedulers for ResNet18 and ResNet34 trained on CIFAR100. Crosses for the same method represent different hyperparameter settings. Each cross represents the mean test accuracy of five random seeds. The best performing model thus far is `TB` combined with `SGDP`.



(a) ResNet18, CIFAR100 (b) ResNet34, CIFAR100 (c) VGG16, CIFAR100 (d) VGG19, CIFAR100

Figure 5: **(Tuning initial learning rate).** Comparing the test accuracy of `TempBalance` (red) and `CAL` baseline (blue) for varying initial learning rate. Our method `TempBalance` outperforms `CAL` for both ResNet and VGG trained on CIFAR100. All results are obtained by running five random seeds.

optimizer [2], such as `AdamW` [11], `AdamP` [41] and `LAMB` [73]. However, we do not find them to provide better results than the `SGD` baseline with cosine annealing (`CAL` in Figure 4).

4.3 Corroborating results and ablation studies.

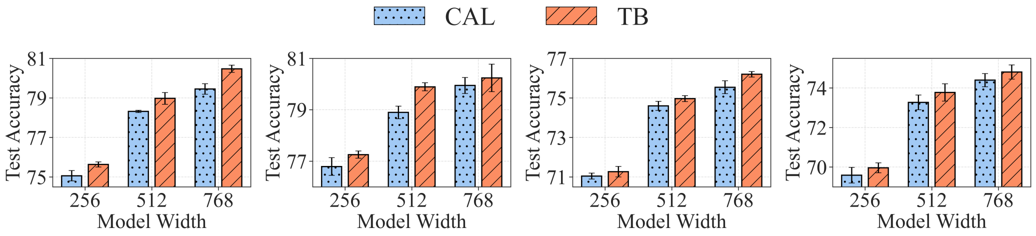
In addition to the main results shown in Figure 3 and Figure 4, we show three ablation studies.

Experiment one: tuning initial learning rate η_0. We train models from scratch using `TempBalance` vs. `CAL` with various initial learning rates. We intend on comparing `TempBalance` and `CAL` baseline when both methods are allowed to search for the optimal hyperparameters. We again use ResNet18, ResNet34, VGG16 and VGG19 as our architectures and show results on CIFAR100. From the results in Figure 5, `TempBalance` achieves a higher test accuracy than `CAL` for both ResNet and VGG.

Experiment two: varying channel width. We view the fraction of model width in Experiment one as "100%" and experiment with models with varied widths in $[50\%, 100\%, 150\%]$. We again use VGG16, VGG19, ResNet18, and ResNet 34 trained on CIFAR100, and we grid search for the optimal learning rate for each width to get best accuracy. From Figure 6, we find that `TempBalance` outperforms the baseline for all widths.

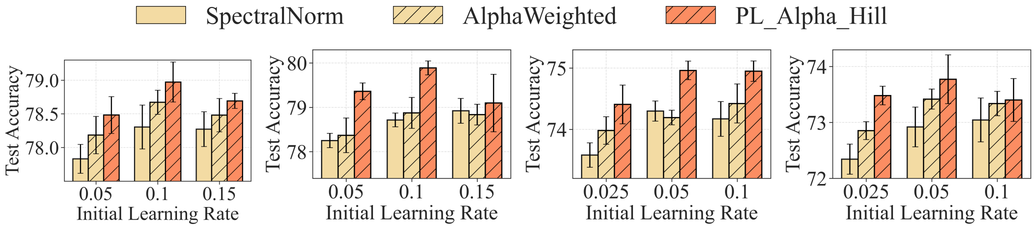
Experiment three: varying HT-SR metric. We use different HT-SR metrics to assign layer-wise learning rates. That is, we replace the layer-wise `PL_Alpha_Hill` in (2) with other HT-SR metrics including `SpectralNorm` and `AlphaWeighted` [30]. Results in Figure 7 show that `PL_Alpha_Hill` achieves the optimal test accuracy.

Visualization results. We further analyze our methods by illustrating the effect of `TempBalance` on regularizing ESDs. See Appendix B.



(a) ResNet18, CIFAR100 (b) ResNet34, CIFAR100 (c) VGG16, CIFAR100 (d) VGG19, CIFAR100

Figure 6: **(Different widths).** Comparing `TempBalance` and the `CAL` baseline for different network widths. All results are obtained by running five random seeds.



(a) ResNet18, CIFAR100 (b) ResNet34, CIFAR100 (c) VGG16, CIFAR100 (d) VGG19, CIFAR100

Figure 7: **(Different HT-SR metrics).** Comparing `PL_Alpha_Hill` with multiple HT-SR metrics. `PL_Alpha_Hill` achieves the best test accuracy among these metrics. All results are obtained by running five random seeds.

5 Conclusion

Our extensive empirical evaluations demonstrate that `TempBalance` offers a straightforward yet effective layer-wise learning rate schedule. Furthermore, our approach for balancing layer-wise temperature confirms the following: (i) HT-SR-motivated metric `PL_Alpha_Hill` helps layers achieve maximal temperature balance during training, exhibits strong correlations with model quality, and yields improved performance during testing. (ii) Temperature balancing is a novel and essential aspect of neural network training, and HT-SR theory provides a strong theoretical support for balancing temperatures. (iii) Layer-wise learning rate schedules are cheap and effective to apply when compared with per-parameter learning rate schedules, and it is useful to study these layer-wise learning rate schedules further. Our method provides insights into the study of layer-wise tuning approaches and load-temperature balancing in deep neural network training, as it serves both as a layer-wise learning rate schedule and an effective regularization technique based on HT-SR metrics.

Limitations and societal impacts. Our paper leaves many future directions to explore, of which we discuss a few below:

- Can HT-SR metrics be extended to parameter-wise learning rate schedules, global learning rate schedules, or other hyperparameters? It would be great to observe how HT-SR can assist in acquiring a comprehensive set of hyperparameter tuning tools.
- Is it possible to accelerate the computation of ESDs and `PL_Alpha_Hill` to achieve a more adaptive learning rate scheduler? Currently, we calculate layer-wise `PL_Alpha_Hill` once per epoch, resulting in a minimal increase in computational complexity. Consider the example of training ResNet18 for 200 epochs on CIFAR100 or TinyImageNet. Calculating layer-wise `PL_Alpha_Hill` takes 1.4 seconds for each epoch, leading to 4.6 minutes in total. Training CIFAR100 on 1 V100 takes 80 minutes, and thus using `TB` increases 6% of training time. Training TinyImageNet on 2 V100 takes 240 minutes, and thus using `TB` increase 2% of training time. However, if we can significantly decrease the expense of computing ESDs, it might enable an optimizer that adjusts the learning rate every few gradient updates.

Our research centers around developing a generic algorithm for optimizing neural networks. Although it can be applied to learning models with adverse applications, we do not see any immediate negative societal impacts stemming from the algorithm itself.

References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[2] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[3] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472, 2017.

[4] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[5] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 2020.

[6] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020.

[7] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

[8] Bharat Singh, Soham De, Yangmuzi Zhang, Thomas Goldstein, and Gavin Taylor. Layer-specific adaptive learning rates for deep networks. In *IEEE 14th International Conference on Machine Learning and Applications*, 2015.

[9] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[10] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[12] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

[13] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, 2015.

[14] Yixiong Chen, Jingxian Li, Hua Jiang, Li Liu, and Chris Ding. Metalr: Layer-wise learning rate based on meta-learning for adaptively fine-tuning medical pre-trained models. *arXiv preprint arXiv:2206.01408*, 2022.

[15] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, 2018.

[16] Youngmin Ro and Jin Young Choi. Autolr: Layer-wise pruning and auto-tuning of learning rates in fine-tuning of deep networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[17] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for ImageNet training. *arXiv preprint arXiv:1708.03888*, 6(12):6, 2017.

[18] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, 2018.

[19] H. S. Seung, H. Sompolinsky, and N. Tishby. Statistical mechanics of learning from examples. *Physical Review A*, 45(8):6056–6091, 1992.

[20] T. L. H. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Rev. Mod. Phys.*, 65(2):499–556, 1993.

[21] D. Haussler, M. Kearns, H. S. Seung, and N. Tishby. Rigorous learning curve bounds from statistical mechanics. *Machine Learning*, 25(2):195–236, 1996.

[22] Andreas Engel and Christian P. L. Van den Broeck. *Statistical mechanics of learning*. Cambridge University Press, New York, NY, USA, 2001.

[23] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations*, 2018.

[24] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

[25] Charles H Martin and Michael W Mahoney. Rethinking generalization requires revisiting old ideas: statistical mechanics approaches and complex learning behavior. Technical Report Preprint: arXiv:1710.09553, 2017.

[26] Yaoqing Yang, Liam Hodgkinson, Ryan Theisen, Joe Zou, Joseph E Gonzalez, Kannan Ramchandran, and Michael W Mahoney. Taxonomizing local versus global structure in neural network loss landscapes. In *Advances in Neural Information Processing Systems*, 2021.

[27] Yefan Zhou, Yaoqing Yang, Arin Chang, and Mahoney W Michael. A three-regime model of network pruning. In *International Conference on Machine Learning*, 2023.

[28] Charles H Martin and Michael W Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *Journal of Machine Learning Research*, 22(165):1–73, 2021.

[29] Charles H Martin and Michael W Mahoney. Traditional and heavy tailed self regularization in neural network models. In *International Conference on Machine Learning*, 2019.

[30] Charles H Martin, Tongsu Serena Peng, and Michael W Mahoney. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12(1):1–13, 2021.

[31] Charles H Martin and Michael W Mahoney. Post-mortem on a deep learning contest: a Simpson's paradox and the complementary roles of scale metrics versus shape metrics. Technical Report Preprint: arXiv:2106.00734, 2021.

[32] Yaoqing Yang, Ryan Theisen, Liam Hodgkinson, Joseph E Gonzalez, Kannan Ramchandran, Charles H Martin, and Michael W Mahoney. Evaluating natural language processing models with generalization metrics that do not need access to any training or testing data. In *proceedings of the 29th SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

[33] Kumar K Agrawal, Arnab Kumar Mondal, Arna Ghosh, and Blake Richards. \alpha-req : Assessing representation quality in self-supervised learning by measuring eigenspectrum decay. In *Advances in Neural Information Processing Systems*, 2022.

[34] Josue Nassar, Piotr Sokol, SueYeon Chung, Kenneth D Harris, and Il Memming Park. On 1/n neural representation and robustness. 2020.

[35] Zeke Xie, Qian-Yuan Tang, Yunfeng Cai, Mingming Sun, and Ping Li. On the power-law hessian spectrums in deep learning. *arXiv preprint arXiv:2201.13011*, 2022.

[36] Bruce M Hill. A simple general approach to inference about the tail of a distribution. *The annals of statistics*, pages 1163–1174, 1975.

[37] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[38] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[39] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[40] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. Powerlaw: a python package for analysis of heavy-tailed distributions. *PloS one*, 9(1):e85777, 2014.

[41] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International Conference on Learning Representations*, 2021.

[42] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, 2019.

[43] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 1982.

[44] Haim Sompolinsky et al. Statistical mechanics of neural networks. *Physics Today*, 41(21): 70–80, 1988.

[45] LM Rasdi Rere, Mohamad Ivan Fanany, and Aniati Murni Arymurthy. Simulated annealing algorithm for deep learning. *Procedia Computer Science*, 72:137–144, 2015.

[46] Adriano Barra and Francesco Guerra. About the ergodic regime in the analogical hopfield neural networks: moments of the partition function. *Journal of mathematical physics*, 49(12): 125217, 2008.

[47] Adriano Barra, Alberto Bernacchia, Enrica Santucci, and Pierluigi Contucci. On the equivalence of hopfield networks and boltzmann machines. *Neural Networks*, 34:1–9, 2012.

[48] Stephen G. Brush. History of the lenz-ising model. *Reviews of Modern Physics*, 39:883–893, 1967.

[49] Marco Baity-Jesi, Levent Sagun, Mario Geiger, Stefano Spigler, Gérard Ben Arous, Chiara Cammarota, Yann LeCun, Matthieu Wyart, and Giulio Biroli. Comparing dynamics: Deep neural networks versus glassy systems. In *International Conference on Machine Learning*, 2018.

[50] Jean Barbier, Florent Krzakala, Nicolas Macris, Léo Miolane, and Lenka Zdeborová. Optimal errors and phase transitions in high-dimensional generalized linear models. *Proceedings of the National Academy of Sciences*, 116(12):5451–5460, 2019.

[51] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. Beyond neural scaling laws: beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems*, 2022.

[52] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2019.

[53] Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M Roy. In search of robust measures of generalization. In *Advances in Neural Information Processing Systems*, 2020.

[54] Peter Bartlett, Dylan Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, 2017.

[55] Huanrui Yang, Xiaoxuan Yang, Neil Zhenqiang Gong, and Yiran Chen. Hero: Hessian-enhanced robust optimization for unifying and improving generalization and quantization performance. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[56] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *IEEE/CVF International Conference on Computer Vision*, 2019.

[57] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-BERT: Hessian based ultra low precision quantization of bert. In *AAAI Conference on Artificial Intelligence*, 2020.

[58] Amartya Sanyal, Philip H. Torr, and Puneet K. Dokania. Stable rank normalization for improved generalization in neural networks and gans. In *International Conference on Learning Representations*, 2020.

[59] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021.

[60] David A McAllester. Some pac-bayesian theorems. In *Proceedings of the eleventh annual conference on Computational learning theory*, 1998.

[61] John Langford and John Shawe-Taylor. Pac-bayes & margins. In *Advances in Neural Information Processing Systems*, 2002.

[62] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Annual Conference on Uncertainty in Artificial Intelligence*, 2017.

[63] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*, 2018.

[64] Xuanzhe Xiao, Zeng Li, Chuanlong Xie, and Fengwei Zhou. Heavy-tailed regularization of weight matrices in deep neural networks. *arXiv preprint arXiv:2304.02911*, 2023.

[65] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. 2009.

[66] Pierre Sermanet, Koray Kavukcuoglu, and Yann LeCun. Traffic signs and pedestrians vision with multi-scale convolutional networks. In *Snowbird Machine Learning Workshop*, 2011.

[67] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE conference on Computer Vision and Pattern Recognition*, 2009.

[68] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[69] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[70] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on Computer Vision and Pattern Recognition*, 2016.

[72] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.

[73] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

[74] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Pathological spectra of the fisher information metric and its variants in deep neural networks. *Neural Computation*, 33:2274–2307, 2019.

[75] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of fisher information in deep neural networks: Mean field approach. In *the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[76] Ka-Veng Yuen. *Bayesian methods for structural dynamics and civil engineering*. John Wiley & Sons, 2010.

[77] Yudi Pawitan. *In all likelihood: statistical modelling and inference using likelihood*. Oxford University Press, 2001.

[78] Bohang Zhang, Du Jiang, Di He, and Liwei Wang. Rethinking lipschitz neural networks for certified l-infinity robustness. *arXiv preprint arXiv:2210.01787*, 2022.

[79] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. PyHessian: Neural networks through the lens of the hessian. In *IEEE International Conference on Big Data (Big Data)*, pages 581–590, 2020.

[80] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[81] Yongqi Du, Di Xie, Shiliang Pu, Robert Qiu, Zhenyu Liao, et al. " lossless" compression of deep neural networks: A high-dimensional neural tangent kernel approach. In *Advances in Neural Information Processing Systems*, 2022.

[82] Zeke Xie, Qian-Yuan Tang, Zheng He, Mingming Sun, and Ping Li. Rethinking the structure of stochastic gradients: Empirical and statistical evidence. *arXiv preprint arXiv:2212.02083*, 2022.

[83] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

[84] Ethan Dyer and Guy Gur-Ari. Asymptotics of wide networks from feynman diagrams. In *International Conference on Learning Representations*, 2020.

[85] Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. In *Proceedings of the National Academy of Sciences*, 2020.

A Heavy-tail phenomena in different DNN matrices are closely related

Recently, several papers have separately studied the HT structures in different types of "important matrices," including Hessian, Fisher Information Matrix (FIM) and covariance matrices [35, 74, 75]. They confirm that when neural networks are well-trained, various matrices have an HT-shaped spectrum. Among these works, there are two major ways to characterize the HT spectrum, namely the HT-shaped ESDs (such as `PL_Alpha`), or HT-shaped decaying eigenvalues [33–35]. Our paper mainly uses the first way of characterizing the HT structure. On the other hand, the second way is to sort eigenvalues from largest to smallest and study the PL phenomena between the ordered eigenvalues and their index. Our experiments show fruitful connections between the PL phenomena manifested in different DNN matrices; if one matrix shows the PL spectrum, the other matrices often show something similar [35]. Thus, It is meaningful to ask why and how the PL phenomena in different prior works correlate.

This section first establishes the connections between input/output covariance matrices, FIM and Hessian in subsection A.1. We find that if one of these matrices shows the PL phenomenon, the other two matrices have a high chance to exhibit a similar PL phenomenon. Then, in subsection A.2, we derive the connection between our metric `PL_Alpha` and the PL exponent on decaying eigenvalues, showing a simple reciprocal relationship between these two.

A.1 Connections between different matrices

Consider a neural network (NN) $f_\theta : \mathbb{R}^d \to \mathbb{R}^C$, where $\theta \in \mathbb{R}^P$ is the vectorized weights, d is the input dimension, and C is the output dimension. When the NN is used for a classifying task, C is also the number of classes. We denote the input data as $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$, and the number of samples is n. We denote the loss function as $L(\theta) = \frac{1}{n}\sum_{i=1}^n l(y_i, f_\theta(x_i))$.

Covariance matrices. We denote the output covariance matrix as $\mathbb{E}[f_\theta(x) f_\theta^\top(x)]$, where the expectation is taken over the input distribution. We tend to consider the following empirical covariance matrix:

$$C(\theta) := \frac{1}{n}\sum_{i=1}^{n} f_\theta(x_i) f_\theta^\top(x_i) \in \mathbb{R}^{C\times C}. \tag{5}$$

Fisher Information Matrices. We denote the (output) FIM as

$$\mathbb{E}[\nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^\top] = \sum_{k=1}^{C} \mathbb{E}[\nabla_\theta f_\theta^{(k)}(x) \nabla_\theta f_\theta^{(k)}(x)^\top], \tag{6}$$

where $f_\theta^{(k)}(x)$ is the k-th entry of the vector function $f(x)$. We also consider the empirical version of the FIM:

$$F(\theta) := \sum_{k=1}^{C} \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta f_\theta^{(k)}(x_i) \nabla_\theta f_\theta^{(k)}(x_i)^\top \in \mathbb{R}^{P\times P}. \tag{7}$$

Note that (7) can be equally written as

$$F(\theta) := \frac{1}{n} \nabla_\theta \tilde{f}_\theta(x) \nabla_\theta \tilde{f}_\theta(x)^\top, \tag{8}$$

where $\nabla_\theta \tilde{f}_\theta(x)$ has the following form:

$$\begin{bmatrix} \frac{\partial f_\theta^{(1)}(x_1)}{\partial \theta_1} \cdots \frac{\partial f_\theta^{(1)}(x_n)}{\partial \theta_1} & \cdots & \frac{\partial f_\theta^{(C)}(x_1)}{\partial \theta_1} \cdots \frac{\partial f_\theta^{(C)}(x_n)}{\partial \theta_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_\theta^{(1)}(x_1)}{\partial \theta_P} \cdots \frac{\partial f_\theta^{(1)}(x_n)}{\partial \theta_P} & \cdots & \frac{\partial f_\theta^{(C)}(x_1)}{\partial \theta_P} \cdots \frac{\partial f_\theta^{(C)}(x_n)}{\partial \theta_P} \end{bmatrix} \in \mathbb{R}^{P\times Cn}.$$

Hessian Matrices. We denote the Hessian as $\mathbb{E}\left[\frac{\partial^2 l(y, f_\theta(x))}{\partial \theta^2}\right]$, and we tend to consider the empirical Hessian Matrices:

$$H(\theta) := \frac{\partial^2 L(\theta)}{\partial \theta^2} \in R^{P\times P}, \tag{9}$$

where $L(\theta)$ is the empirical loss function $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, f_\theta(x_i))$.

Hessian and FIM are equivalent under certain conditions. FIM can be defined in alternative ways different from (6). For instance, from classic statistical knowledge, we have the standard FIM (sFIM) in the following form:

$$sFIM := \mathbb{E}[\nabla_\theta \log P(y|x;\theta) \nabla_\theta \log P(y|x;\theta)^T], \tag{10}$$

where $P(y|x;\theta)$ represents the likelihood; after simple derivations, one can show that sFIM also has the following form [76, 77]:

$$sFIM = -\mathbb{E}\left[\frac{\partial^2 \log P(y|x;\theta)}{\partial\theta^2}\right]. \tag{11}$$

Therefore, when the loss function is defined as the negative log-likelihood, the sFIM in (11) is equivalent to Hessian defined in (9).

Why is the FIM defined in (6) **equivalent to** (10)**.** Back to deep learning, FIM is often defined as (6). It is thus meaningful to derive the equivalence between these two forms. Suppose $P(y|x;\theta)$ here means the conditional probability distribution of output y given input data x. If $P(y|x;\theta)$ is assumed to take the following form:

$$P(y|x;\theta) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\|y - f_\theta(x)\|^2\right), \tag{12}$$

then the MSE estimator $min_\theta \ \frac{1}{2}\|y - f_\theta(x)\|^2$ is equivalent to the maximum likelihood estimation of $P(y|x;\theta)$. Then, plugging (12) into (10), we have:

$$sFIM_{mse} = \mathbb{E}[\|y - f_\theta(x)\|^2 \nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^T]. \tag{13}$$

We now expand $sFIM_{mse}$ by the definition of expectation, and we have the following [74]:

$$\begin{aligned}
sFIM_{mse} &= \int_{\mathbb{R}}\int_{\mathbb{R}} \|y - f_\theta(x)\|^2 \nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^T p(x,y;\theta)dydx && (14)\\
&= \int_{\mathbb{R}}\int_{\mathbb{R}} \|y - f_\theta(x)\|^2 \nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^T P(y|x;\theta)q(x)dydx && (15)\\
&= \int_{\mathbb{R}}\left[\int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}}\|y - f_\theta(x)\|^2 \exp\left(-\frac{1}{2}\|y - f_\theta(x)\|^2\right)dy\right] \nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^T q(x)dx && (16)\\
&= \int_{\mathbb{R}} \nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^T q(x)dx && (17)\\
&= \mathbb{E}[\nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^T], && (18)
\end{aligned}$$

where (14) follows from the definition of expectation, $q(x)$ is input distribution, and (17) holds because the integral of y in the brackets [] equals 1 due to the property of Gamma function $\Gamma(\cdot)$.

Therefore, from (18), we find that $sFIM_{mse}$ is just equal to FIM defined in (6). Also, plugging (12) into $\mathbb{E}\left[\frac{\partial^2 logP(y|x;\theta)}{\partial\theta^2}\right]$ and taking the loss function $L(\theta)$ as the mean-square loss, we will again find $\mathbb{E}\left[\frac{\partial^2 logP(y|x;\theta)}{\partial\theta^2}\right]$ is equal to $H(\theta)$. Therefore, jointly considering (11), we can see that FIM is equal to Hessian $H(\theta)$.

PL in the covariance matrix and PL in Hessian are tightly correlated. Next, we consider the relationship between the covariance matrix and Hessian. Suppose the NN function f_θ is a Lipchitz function [78]. Then, it can be seen that the covariance matrix (5) may be controlled and estimated by FIM defined in (6), which is equivalent to be controlled by Hessian.

Although deriving an exact equivalent between these two can be hard, we numerically show that the PL in one matrix informs the PL in the other. To visualize their relationship in the presence of PL, we train a simple MLP with one hidden layer and 2000 neurons for 50 epochs. We leverage the spectral regularization from Nassar et al. [34] to make the output covariance matrix exhibit a PL spectrum. Meanwhile, we calculate the top eigenvalues of covariance and hessian [79], fit the PL exponent s for

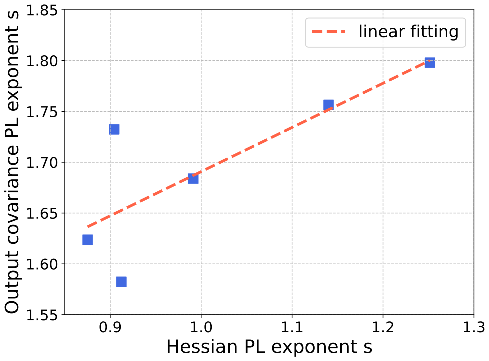


Figure 8: We train a MLP for 50 epochs and fit PL exponent s for both the output covariance and the Hessian. For models trained with epochs [1,10,20,30,40,50], we see their PL exponents s show a strong correlation.

each matrix, and compare the PL exponents against each other. More specifically, we take trained NNs from epochs [1,10,20,30,40,50] and plot the Hessian PL exponent s versus the output covariance PL exponent s. From the results shown in figure 8, we can see that their PL exponent s shows a strong correlation, which supports our claim that the PL phenomena in one matrix can inform the other.

Connections to the NTK matrix. Interestingly, if we ignore the constant in (8) and switch the two matrices multiplied together, we obtain $\nabla_\theta \tilde{f}_\theta(x)^T \nabla_\theta \tilde{f}_\theta(x)$. This matrix is equal to Neural Tangent Kernel(NTK) [80], which is a kernel used to approximate the deep neural network when NN's width is infinite. We thus conjecture that NTK should show PL when the NN is well trained [81]. Indeed, Karakida et al. [74] and Karakida et al. [75] study the eigenvalues of NTK, showing a PL trend. Some other work on stochastic gradient [82] claim that the so-called "stochastic gradient matrix" (which is similar to the NTK matrix) shows a PL spectrum as well, which matches our expectations. Also, Lewkowycz et al. [83], Dyer and Gur-Ari [84] show that the eigenvalues of NTK are similar to those in the Hessian, which again meets our expectation because the Hessian tends to be PL when neural networks are well-trained [35].

In summary, the derivations above indicate that different "important matrices" are tightly correlated to each other in terms of the PL trends: if one matrix shows a PL spectrum, there is a high chance that the other ones show something similar.

A.2 Connections between PL in ESD and PL in decaying eigenvalues

Next, we derive the connection between our metric `PL_Alpha` and the exponent of PL distribution on decaying eigenvalues. Take the covariance matrix (5) as an instance. According to Nassar et al. [34], the HT phenomenon in the output covariance matrix is similar to the layer-wise covariance matrices. Thus, without the loss of generality, we can consider the case when there is only one layer in the neural network. We assume the weight matrix L is in $\mathbf{R}^{N\times M}$. According to prior works, when L is well-trained, the ESD follows a PL distribution:

$$p(\lambda) = \frac{1}{H}\lambda^{-\alpha}, \quad \lambda_{min} < \lambda < \lambda_{max}. \tag{19}$$

Here, H is a normalizing constant, and α is the PL exponent.

Another way to characterize the PL phenomenon is to consider eigenvalues directly following a PL series. For example, Xie et al. [35] show that the decaying eigenvalues follows the following PL series:

$$\lambda_k = \lambda_1 k^{-s}, k = 1, 2, \cdots, M, \tag{20}$$

where λ_1 is the same as λ_{max} used in the main paper.

Now, we will analytically and empirically show that these two ways of characterizing PL are essentially equivalent. Furthermore, the two PL coefficients satisfy $s = \frac{1}{\alpha - 1}$.

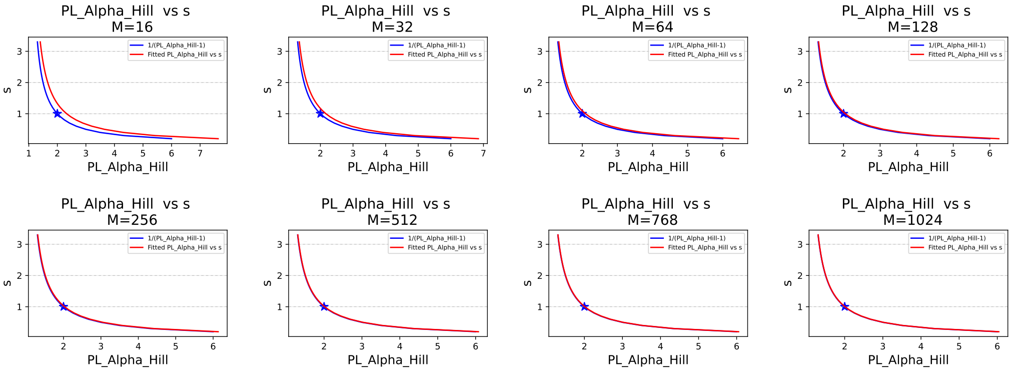


Figure 9: We show the connection between `PL_Alpha`, estimated by `PL_Alpha_Hill`, and the PL exponent of the decaying eigenvalues (denoted as s) satisfy $s = \frac{1}{\texttt{PL\_Alpha\_Hill}-1}$. Results are shown for different matrix size M. In particular, we see that $\texttt{PL\_Alpha} = 2$ [28] is equivalent to $s = 1$ [33] in the linear case.

An analytical way to show that $s = \frac{1}{\alpha-1}$**.** The derivation is actually quite simple. Consider the case that $\lambda_k = \lambda_1 k^{-s}$ (i.e., (20) holds), and suppose Λ is a random variable distributed according to the empirical distribution from these eigenvalues $\lambda_k = \lambda_1 k^{-s}$. Now, from (20), we easily see that the distribution function takes the following form:

$$\mathbb{P}(\Lambda > \lambda_1 k^{-s}) = \frac{k}{M}. \tag{21}$$

By changing variables $\lambda_1 k^{-s} = \lambda$, we get the cumulative distribution function of Λ:

$$\mathbb{P}(\Lambda > \lambda) \sim \lambda^{-\frac{1}{s}}. \tag{22}$$

After that, we take the derivative with respect to λ, and we get the ESD:

$$p(\lambda) \sim \lambda^{-(\frac{1}{s}+1)}. \tag{23}$$

In other words, we have $\lambda^{-(\frac{1}{s}+1)} = \lambda^{-\alpha}$, which means $s = \frac{1}{\alpha-1}$.

An empirical way to show that $s = \frac{1}{\alpha-1}$**.** We consider matrices of size $M \times M$, where we choose M in $[16, 32, 64, 128, 256, 512, 768, 1024]$ and assign the parameters such that the decaying eigenvalues obey the formula $\lambda_1 k^{-s}$, for s in $[0.2, 0.3, 0.4, \cdots, 3.2]$. Then, we fit the ESD and get our estimate `PL_Alpha_Hill`. We plot the relationship between `PL_Alpha_Hill` and s in figure9. From figure 9, we find that the connection between α and s shows a good fit with the formula $s = \frac{1}{\alpha-1}$. With increasing matrix size M, the fitting becomes increasingly accurate.

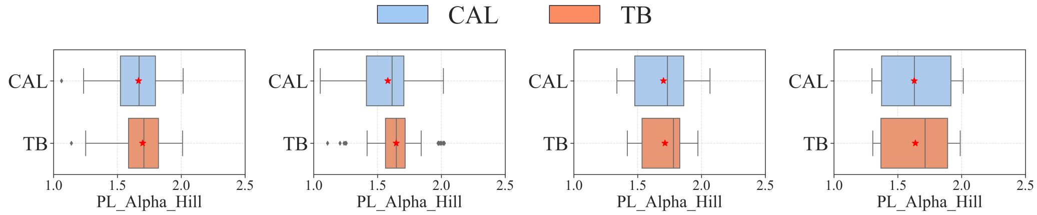
When $s = \frac{1}{\alpha-1}$**,** $s = 1$ **corresponds to** $\alpha = 2$**.** Some prior works Nassar et al. [34], Xie et al. [35], Bartlett et al. [85] measure the HT phenomena from the perspective of decaying eigenvalues with PL exponent s, and they show either theoretically or empirically that $s = 1$ is the *optimal* exponent. Now that we have $s = \frac{1}{\alpha-1}$ in the linear case, and from the theory of NTK[80], the infinite wide neural network is approximated as a linear model, we tend to believe that $\alpha = 2$ satisfies a similar property. Indeed, one of the main contributions of Martin and Mahoney [28] is to establish different HT families of ESDs, and 2 is believed to be the boundary between "moderately HT" and "very HT." Martin and Mahoney [28] further argue that the optimal exponent for `PL_Alpha` is in the range [2,4]. Combining the perspective from Nassar et al. [34], Xie et al. [35], Bartlett et al. [85] and those from Martin and Mahoney [28], it is reasonable to believe that the optimal exponent for `PL_Alpha` is around 2. When `PL_Alpha` is much higher or lower than two, the NN probably has some issue in training. Although we argued in the main paper that the "absolute value" of `PL_Alpha` is unimportant in implementing our `TempBalance` algorithm, it is, however, helpful to have an "optimal" `PL_Alpha` value to test if our algorithm actually works in controlling the ESDs. We will show visualization results in Appendix B that `TempBalance` leads to a better distribution of our estimated `PL_Alpha_Hill`.

B Visualization results: how does `TempBalance` control ESDs

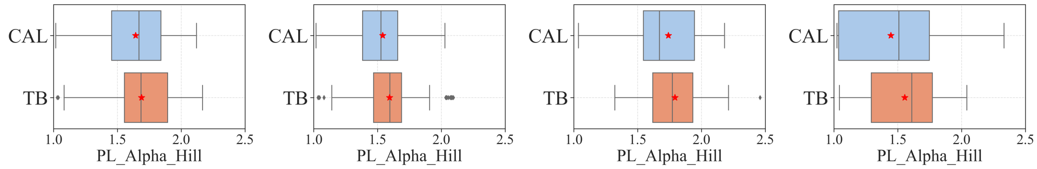
In this section, we demonstrate that the proposed method, `TempBalance`, effectively controls the shape of the empirical spectral densities (ESDs), resulting in a more favorable distribution of `PL_Alpha_Hill` among the layers of neural networks (NNs) compared to the baseline method `CAL`. This observation elucidates the superior performance of `TempBalance` over `CAL` in our main experiment, as presented in Section 4.2.

We evaluate the models reported in Figure 3. For each individual NN, we compute and aggregate `PL_Alpha_Hill` values across all layers, excluding the first and last layers that have an extremely small number of eigenvalues and thus cause inaccurate `PL_Alpha_Hill` estimation. We aggregate the `PL_Alpha_Hill` values from five models trained using different random seeds for each method. Figure 10 shows the distribution of `PL_Alpha_Hill` of `TempBalance` and the baseline `CAL`. Comparing `TempBalance` with `CAL`, we see that `TempBalance` consistently yields a more concentrated distribution. Furthermore, `TempBalance` causes the median and mean of the distribution to approach 2 (shown in each subplot respectively as the middle vertical line and the red star). The value 2 represents the theoretically optimal `PL_Alpha_Hill` value, as we have justified in Appendix A.

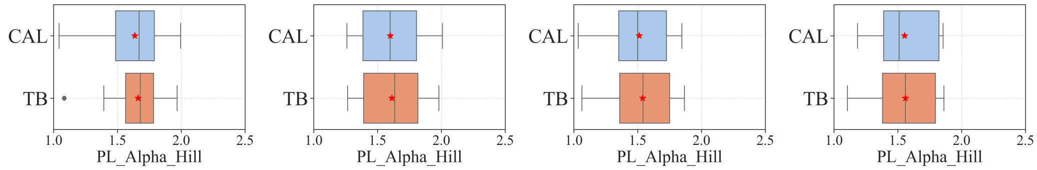
Next, in Figure 11, we group the models into different subgroups based on their architectures and/or datasets, aggregating the `PL_Alpha_Hill` values and comparing the distributions of the two methods `TempBalance` and `CAL`. Once again, we observe that `TempBalance` results in a more concentrated distribution, with a larger number of samples (layers) having `PL_Alpha_Hill` values closer to 2.



(a) ResNet18, CIFAR100 (b) ResNet34, CIFAR100 (c) VGG16, CIFAR100 (d) VGG19, CIFAR100

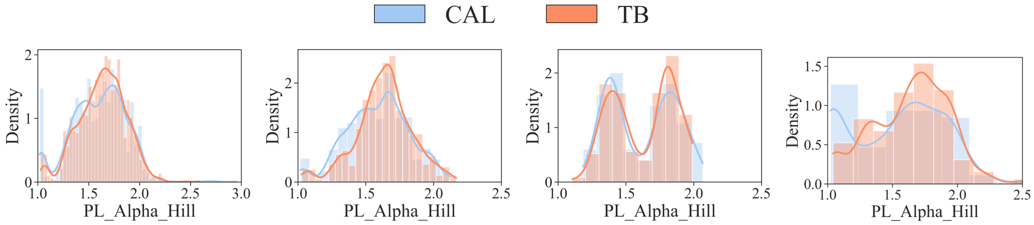


(e) ResNet18, TIN (f) ResNet34, TIN (g) WRN16-8, TIN (h) WRN28-6, TIN

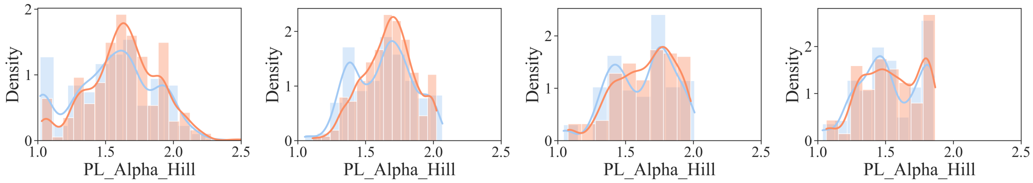


(i) ResNet18, CIFAR10 (j) VGG16, CIFAR10 (k) ResNet18, SVHN (l) VGG16, SVHN

Figure 10: Comparing the distribution of `PL_Alpha_Hill` of NNs trained by our method `TempBalance` (TB) and `CAL`. The mean of each distribution is indicated by a red star marker. Each distribution aggregates the `PL_Alpha_Hill` values from models trained using five different random seeds. Across all experiments, our method `TempBalance` consistently yields a more concentrated distribution, resulting in the mean and median approaching the theoretically optimal `PL_Alpha_Hill` value of 2, as supported in Appendix A.



(a) Total (b) ResNet (c) VGG (d) WRN



(e) TIN (f) CIFAR100 (g) CIFAR10 (h) SVHN

Figure 11: Comparing our method `TempBalance` (TB) to `CAL` in terms of the distribution of `PL_Alpha_Hill` of aggregating NNs into different architectures and datasets. Each distribution aggregates the `PL_Alpha_Hill` of models trained with five random seeds. Across all subgroups, our method `TempBalance` consistently exhibits a more concentrated distribution, accompanied by a higher number of layers approaching a `PL_Alpha_Hill` value close to 2. This value of 2 corresponds to the theoretically optimal `PL_Alpha_Hill` value, as justified in Appendix A.

C Hyperparameter settings for reproducing our results

In this section, we report all hyperparameters, random seeds and all numerical values of experimental results shown in the main paper (in Section 4).

First, we report the common hyperparameters shared by all the experiments: the default optimizer is SGD, trained with batch size 128, number of training epochs 200, weight decay 5e-4, and momentum 0.9. The default HT-SR metric used in `TempBalance` is `PL_Alpha_Hill`. For each experimental setting, we use five random seeds, which are always 43, 37, 13, 51, 71, and we report the mean and standard deviation of the test accuracy across these seeds.

First, Table 1 reports the details of experiments shown in Figure 3. We carefully tune the initial learning rate η_0 and λ_{sr} for the two baseline methods `CAL` and `SNR`. Then, Table 2 reports the detailed hyperparameter settings of the experiments shown in Figure 4. We again carefully tune the hyperparameters of various baseline optimizers and schedulers, as specified in their papers. Finally, Table 3, Table 4 and Table 5 respectively report the details of the experiments shown in Figure 5, Figure 6 and Figure 7.

Table 1: Parameter settings of the experiment reported in Section 4.2 Figure 3. The hyperparameter in bold is the best hyperparameter selection reported in the main paper. The five random seeds for each setting are {43, 37, 13, 51, 71}, and the means and standard deviations of the test accuracy among the five seeds are reported.

Index	Dataset	Model	Method	Initial learning rate η_0	λ_{sr}	Test Acc (best hyperparam.)	scaling ratio (s_1, s_2)
0	CIFAR100	ResNet18	`CAL`	0.05, **0.1**, 0.15	-	78.31 ± 0.05	-
1		ResNet18	`SNR`	0.1	0.001, 0.005, **0.01**, 0.015	78.65 ± 0.29	-
2		ResNet18	`TB`	0.1	-	78.97 ± 0.29	(0.5, 1.5)
3		ResNet18	`TB + SNR`	0.1	0.001	79.06 ± 0.32	(0.6, 1.4)
4		ResNet34	`CAL`	**0.05**, 0.1, 0.15	-	78.98 ± 0.14	-
5		ResNet34	`SNR`	0.1	0.001, 0.005, 0.01, **0.015**	79.97 ± 0.21	-
6		ResNet34	`TB`	0.1	-	79.89 ± 0.15	(0.5, 1.5)
7		ResNet34	`TB + SNR`	0.1	0.005	80.09 ± 0.35	(0.6, 1.4)
8		VGG16	`CAL`	0.025, **0.05**, 0.1	-	74.59 ± 0.23	-
9		VGG16	`SNR`	0.05	0.001, **0.005**, 0.01, 0.015	74.80 ± 0.28	-
10		VGG16	`TB`	0.05	-	74.96 ± 0.15	(0.5, 1.5)
11		VGG16	`TB + SNR`	0.05	0.005	75.52 ± 0.46	(0.6, 1.4)
12		VGG19	`CAL`	0.025, **0.05**, 0.1	-	73.26 ± 0.37	-
13		VGG19	`SNR`	0.05	0.001, 0.005, **0.01**, 0.015	74.37 ± 0.16	-
14		VGG19	`TB`	0.05	-	73.77 ± 0.43	(0.5, 1.5)
15		VGG19	`TB + SNR`	0.05	0.01	74.74 ± 0.10	(0.5, 1.5)
16	TinyImageNet	ResNet18	`CAL`	0.05, **0.1**, 0.15	-	66.25 ± 0.17	-
17		ResNet18	`SNR`	0.1	0.001, 0.005, **0.01**, 0.015	66.20 ± 0.22	-
18		ResNet18	`TB`	0.1	-	66.77 ± 0.25	(0.6, 1.4)
19		ResNet18	`TB + SNR`	0.1	0.001	66.86 ± 0.22	(0.6, 1.4)
20		ResNet34	`CAL`	0.05, **0.1**, 0.15	-	68.19 ± 0.16	-
21		ResNet34	`SNR`	0.1	0.001, 0.005, **0.01**, 0.015	68.69 ± 0.13	-
22		ResNet34	`TB`	0.1	-	69.12 ± 0.16	(0.6, 1.4)
23		ResNet34	`TB + SNR`	0.1	0.001	69.27 ± 0.21	(0.6, 1.4)
24		WRN16-8	`CAL`	0.05, **0.1**, 0.15	-	63.67 ± 0.09	-
25		WRN16-8	`SNR`	0.1	0.00005, **0.0001**, 0.001	63.98 ± 0.23	-
26		WRN16-8	`TB`	0.1	-	64.09 ± 0.17	(0.6, 1.4)
27		WRN16-8	`TB + SNR`	0.1	0.0001	64.08 ± 0.07	(0.6, 1.4)
28		WRN28-6	`CAL`	**0.05**, 0.1, 0.15	-	65.88 ± 0.20	-
29		WRN28-6	`SNR`	0.1	0.00005, **0.0001**, 0.001	66.09 ± 0.25	-
30		WRN28-6	`TB`	0.1	-	66.58 ± 0.23	(0.6, 1.4)
31		WRN28-6	`TB + SNR`	0.1	0.0001	66.79 ± 0.25	(0.6, 1.4)
32	CIFAR10	ResNet18	`CAL`	0.05, **0.1**, 0.15	-	95.53 ± 0.12	-
33		ResNet18	`SNR`	0.1	**0.001**, 0.005, 0.01, 0.015	95.57 ± 0.06	-
34		ResNet18	`TB`	0.1	-	95.63 ± 0.08	(0.5, 1.5)
35		ResNet18	`TB + SNR`	0.1	0.001	95.66 ± 0.09	(0.6, 1.4)
36		VGG16	`CAL`	0.025, 0.05, **0.1**	-	93.98 ± 0.12	-
37		VGG16	`SNR`	0.05	0.001, **0.005**, 0.01, 0.015	94.04 ± 0.07	-
38		VGG16	`TB`	0.05	-	94.14 ± 0.06	(0.5, 1.5)
39		VGG16	`TB + SNR`	0.05	0.005	94.26 ± 0.10	(0.6, 1.4)
40	SVHN	ResNet18	`CAL`	0.05, **0.1**, 0.15	-	96.59 ± 0.08	-
41		ResNet18	`SNR`	0.1	0.001, **0.005**, 0.015, 0.01	96.65 ± 0.12	-
42		ResNet18	`TB`	0.1	-	96.63 ± 0.06	(0.5, 1.5)
43		ResNet18	`TB + SNR`	0.1	0.01	96.67 ± 0.09	(0.6, 1.4)
44		VGG16	`CAL`	0.025, **0.05**, 0.1	-	96.28 ± 0.04	-
45		VGG16	`SNR`	0.05	0.001, 0.005, **0.015**, 0.01	96.32 ± 0.07	-
46		VGG16	`TB`	0.05	-	96.33 ± 0.06	(0.5, 1.5)
47		VGG16	`TB + SNR`	0.05	0.005	96.40 ± 0.08	(0.6, 1.4)

Table 2: Parameter settings of the experiment reported in Section 4.2 Figure 4. The hyperparameter in bold is the best hyperparameter selection reported in the main paper. The five random seeds for each setting are {43, 37, 13, 51, 71}, and the means and standard deviations of the test accuracy among the five seeds are reported.

Index	Dataset	Model	Method	Initial learning rate η_0	SGDR (T_0, T_{mul})	Lookahead k	Lookahead α	Test Acc (best hyperparams.)	scaling ratio (s_1, s_2)
0	CIFAR100	ResNet18	`CAL`	0.05, **0.1**, 0.15	-	-	-	78.31 ± 0.05	-
1		ResNet18	`SGDR`	0.05, **0.1**, 0.15	**(100,1)**, (10, 2),(1, 2)	-	-	77.69 ± 0.20	-
2		ResNet18	`LARS`	26, **28**, 30, 32, 34	-	-	-	78.44 ± 0.12	-
3		ResNet18	`Lookahead`	0.05, **0.1**, 0.15	-	**10**, 5	**0.8**, 0.5	78.46 ± 0.18	-
4		ResNet18	`SGDP`	0.01, 0.05, **0.1**, 0.15, 0.2	-	-	-	78.74 ± 0.11	-
5		ResNet18	`TB`	0.05, **0.1**, 0.15	-	-	-	78.97 ± 0.29	(0.5, 1.5)
6		ResNet18	`TB + SGDP`	0.05, **0.1**, 0.15	-	-	-	79.13 ± 0.15	(0.5, 1.5)
7		ResNet34	`CAL`	**0.05**, 0.1, 0.15	-	-	-	78.98 ± 0.14	-
8		ResNet34	`SGDR`	**0.05**, 0.1, 0.15	**(100,1)**, (10, 2), (1, 2)	-	-	78.61 ± 0.20	-
9		ResNet34	`LARS`	26, 28, 30, **32**, 34	-	-	-	78.94 ± 0.19	-
10		ResNet34	`Lookahead`	0.05, 0.1, **0.15**	-	**10**, 5	**0.8**, 0.5	79.19 ± 0.12	-
11		ResNet34	`SGDP`	0.01, 0.05, **0.1**, 0.15, 0.2	-	-	-	79.34 ± 0.21	-
12		ResNet34	`TB`	0.05, **0.1**, 0.15	-	-	-	79.89 ± 0.15	(0.5, 1.5)
13		ResNet34	`TB + SGDP`	0.05, **0.1**, 0.15	-	-	-	79.94 ± 0.30	(0.5, 1.5)

Table 3: Parameter settings of the experiment reported in Section 4.3 Figure 5. The five random seeds for each setting are {43, 37, 13, 51, 71}, and the means and standard deviations of the test accuracy among the five seeds are reported.

Index	Dataset	Model	Method	Initial learning rate η_0	Test Acc	scaling ratio (s_1, s_2)
0	CIFAR100	ResNet18	`CAL`	0.05, 0.1, 0.15	78.08 ± 0.19, 78.31 ± 0.05, 77.72 ± 0.44	-
1		ResNet18	`TB`	0.05, 0.1, 0.15	78.48 ± 0.27, 78.97 ± 0.29, 78.69 ± 0.11	(0.5, 1.5)
2		ResNet34	`CAL`	0.05, 0.1, 0.15	78.98 ± 0.14, 78.89 ± 0.24, 78.51 ± 0.34	-
3		ResNet34	`TB`	0.05, 0.1, 0.15	79.36 ± 0.18, 79.89 ± 0.15, 79.09 ± 0.64	(0.5, 1.5)
4		VGG16	`CAL`	0.025, 0.05, 0.1	73.96 ± 0.27, 74.59 ± 0.23, 74.46 ± 0.12	-
5		VGG16	`TB`	0.025, 0.05, 0.1	74.40 ± 0.31, 74.96 ± 0.15, 74.94 ± 0.16	(0.5, 1.5)
6		VGG19	`CAL`	0.025, 0.05, 0.1	72.57 ± 0.45, 73.26 ± 0.37, 72.98 ± 0.16	-
7		VGG19	`TB`	0.025, 0.05, 0.1	73.47 ± 0.16, 73.77 ± 0.43, 73.40 ± 0.38	(0.5, 1.5)

Table 4: Parameter settings of the experiment reported in Section 4.3 Figure 6. The five random seeds for each setting are {43, 37, 13, 51, 71}, and the means and standard deviations of the test accuracy among the five seeds are reported.

Index	Dataset	Model	Method	Initial learning rate η_0	Width	Test Acc	scaling ratio (s_1, s_2)
0	CIFAR100	ResNet18	`CAL`	0.1	256, 512, 768	75.05 ± 0.26, 78.31 ± 0.05, 79.44 ± 0.26	-
1		ResNet18	`TB`	0.1	256, 512, 768	75.63 ± 0.12, 78.97 ± 0.29, 80.47 ± 0.18	(0.5, 1.5)
2		ResNet34	`CAL`	0.1	256, 512, 768	76.79 ± 0.34, 78.89 ± 0.24, 79.94 ± 0.31	-
3		ResNet34	`TB`	0.1	256, 512, 768	77.25 ± 0.14, 79.89 ± 0.15, 80.23 ± 0.53	(0.5, 1.5)
4		VGG16	`CAL`	0.05	256, 512, 768	71.04 ± 0.14, 74.59 ± 0.23, 75.53 ± 0.32	-
5		VGG16	`TB`	0.05	256, 512, 768	71.26 ± 0.26, 74.96 ± 0.15, 76.19 ± 0.14	(0.5, 1.5)
6		VGG19	`CAL`	0.05	256, 512, 768	69.58 ± 0.39, 73.26 ± 0.37, 74.39 ± 0.33	-
7		VGG19	`TB`	0.05	256, 512, 768	69.96 ± 0.25, 73.77 ± 0.43, 74.80 ± 0.35	(0.5, 1.5)

Table 5: Parameter settings of the experiment reported in Section 4.3 Figure 7. The five random seeds for each setting are {43, 37, 13, 51, 71}, and the means and standard deviations of the test accuracy among the five seeds are reported.

Index	Dataset	Model	Method	HT-SR Metric	Initial learning rate η_0	Test Acc	scaling ratio (s_1, s_2)
0	CIFAR100	ResNet18	`TB`	`SpectralNorm`	0.05, 0.1, 0.15	77.83 ± 0.21, 78.30 ± 0.32, 78.27 ± 0.25	(0.5, 1.5)
1		ResNet18	`TB`	`AlphaWeighted`	0.05, 0.1, 0.15	78.18 ± 0.27, 78.67 ± 0.17, 78.48 ± 0.24	(0.5, 1.5)
1		ResNet18	`TB`	`PL_Alpha_Hill`	0.05, 0.1, 0.15	78.48 ± 0.27, 78.97 ± 0.29, 78.69 ± 0.11	(0.5, 1.5)
2		ResNet34	`TB`	`SpectralNorm`	0.05, 0.1, 0.15	78.25 ± 0.16, 78.71 ± 0.15, 78.92 ± 0.28	(0.5, 1.5)
3		ResNet34	`TB`	`AlphaWeighted`	0.05, 0.1, 0.15	78.36 ± 0.39, 78.87 ± 0.34, 78.83 ± 0.23	(0.5, 1.5)
3		ResNet34	`TB`	`PL_Alpha_Hill`	0.05, 0.1, 0.15	79.36 ± 0.18, 79.89 ± 0.15, 79.09 ± 0.64	(0.5, 1.5)
4		VGG16	`TB`	`SpectralNorm`	0.025, 0.05, 0.1	73.58 ± 0.19, 74.29 ± 0.16, 74.17 ± 0.28	(0.5, 1.5)
5		VGG16	`TB`	`AlphaWeighted`	0.025, 0.05, 0.1	73.97 ± 0.22, 74.19 ± 0.11, 74.42 ± 0.31	(0.5, 1.5)
5		VGG16	`TB`	`PL_Alpha_Hill`	0.025, 0.05, 0.1	74.40 ± 0.31, 74.96 ± 0.15, 74.94 ± 0.16	(0.5, 1.5)
6		VGG19	`TB`	`SpectralNorm`	0.025, 0.05, 0.1	72.34 ± 0.26, 72.91 ± 0.35, 73.04 ± 0.39	(0.5, 1.5)
7		VGG19	`TB`	`AlphaWeighted`	0.025, 0.05, 0.1	72.85 ± 0.16, 73.41 ± 0.17, 73.33 ± 0.21	(0.5, 1.5)
7		VGG19	`TB`	`PL_Alpha_Hill`	0.025, 0.05, 0.1	73.47 ± 0.16, 73.77 ± 0.43, 73.40 ± 0.38	(0.5, 1.5)