LoRA+: Efficient Low Rank Adaptation of Large Models

Soufiane Hayou [*1] **Nikhil Ghosh** [*2] **Bin Yu** [2]

Abstract

In this paper, we show that Low Rank Adaptation (LoRA) as originally introduced in (Hu et al., 2021) leads to suboptimal finetuning of models with large width (embedding dimension). This is due to the fact that adapter matrices A and B in LoRA are updated with the same learning rate. Using scaling arguments for large width networks, we demonstrate that using the same learning rate for A and B does not allow efficient feature learning. We then show that this suboptimality of LoRA can be corrected simply by setting different learning rates for the LoRA adapter matrices A and B with a well-chosen fixed ratio. We call this proposed algorithm LoRA$+$. In our extensive experiments, LoRA$+$ improves performance ($1\% - 2\%$ improvements) and finetuning speed (up to $\sim$ 2X SpeedUp), at the same computational cost as LoRA.

1. Introduction

State-of-the-art (SOTA) deep learning models all share a common characteristic: they all have an extremely large number of parameters (10's if not 100's of billions parameters). Currently, only a few industry labs can pretrain large language models due to their high training cost. However, many pretrained models are accessible either through an API (GPT4, (OpenAI, 2023)) or through open-source platforms (Llama, (Touvron et al., 2023)). Most practitioners are interested in using such models for specific tasks and want to *adapt* these models to a new, generally smaller task. This procedure is known as *finetuning*, where one adjusts the weights of the pretrained model to improve performance on the new task. However, due to the size of SOTA models, adapting to down-stream tasks with full finetuning (finetuning all model parameters) is computationally infeasible as it requires modifying the weights of the pretrained models using gradient methods which is a costly process. Besides, a model that has already learned generally useful representations during pretraining would not require in-principle significant adaptation of all parameters. With this intuition, researchers have proposed a variety of resource-efficient finetuning methods which typically freeze the pretrained weights and tune only a small set of newly inserted parameters. Such methods include prompt tuning (Lester et al., 2021) where a "soft prompt" is learned and appended to the input, the adapters method (Houlsby et al., 2019) where lightweight "adapter" layers are inserted and trained, and $(IA)^3$ (Liu et al., 2022) where activation vectors are modified with learned scalings. Another resource-efficient method is known as *Low Rank Adaptation* (Hu et al., 2021), or simply LoRA. In LoRA finetuning, only a low rank matrix, called an *adapter*, that is added to the pretrained weights is trainable. The training can be done with any optimizer and in practice a common choice is Adam (Kingma and Ba, 2014). Since the trained adapter is low-rank, this effectively reduces the number of trainable parameters in the fine-tuning process, significantly decreasing the training cost. On many tasks such as instruction finetuning, LoRA has been shown to achieve comparable or better performance compared with full-finetuning (Wang et al., 2023; Liu et al., 2023), although on complicated, long form generation tasks, it is not always as performant. The impressive performance and the computational savings of LoRA have contributed to it becoming an industry standard finetuning method.

Efficient use of LoRA requires a careful choice of hyperparameters: the rank and the learning rate. While some theoretical guidelines on the choice of the rank in LoRA exist in the literature (see e.g. Zeng and Lee (2023)), there are no principled guidelines on how to set the learning rate, apart from common choices of order $1e$-4.

Related Work. Dettmers et al. (2023) introduced a quantized version of LoRA (or QLoRA), which further reduces computation costs by quantizing pretrained weights down to as few as four bits. Using QLoRA enables fine-tuning Llama-65b (Touvron et al., 2023), on a single consumer GPU while achieving competitive performance with full-finetuning. To further improve LoRA training with quantization, Li et al. (2023) introduced a new

[*] Equal contribution [1] Simons Institute, UC Berkeley [2] Department of Statistics, UC Berkeley. Correspondence to: Soufiane Hayou <hayou@berkeley.edu>, Nikhil Ghosh <nikhil_ghosh@berkeley.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

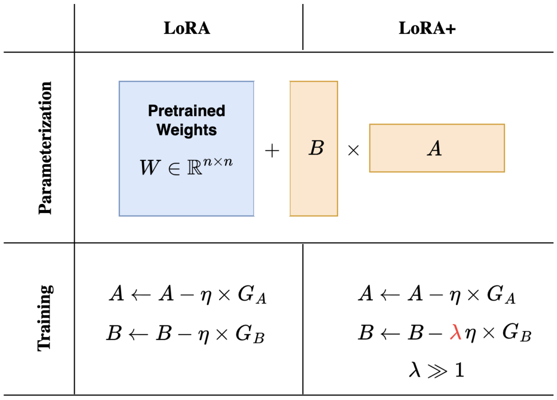

LoRA
LoRA+
Parameterization
Pretrained Weights
$W \in \mathbb{R}^{n\times n}$
$+$
B
$\times$
A
Training
$A \leftarrow A - \eta \times G_A$
$B \leftarrow B - \eta \times G_B$
$A \leftarrow A - \eta \times G_A$
$B \leftarrow B - \lambda\eta \times G_B$
$\lambda \gg 1$

Figure 1. The key difference between standard LoRA and LoRA+ is in how learning rates are set (the matrices G_A and G_B are 'effective' gradients from AdamW) With standard LoRA, the learning rate is the same for A and B, which provably leads to suboptimal learning when embedding dimension is large. In LoRA+, we set the learning rate of B to be $\lambda\times$ that of A, where $\lambda \gg 1$ is fixed. We later provide guidelines on how to set λ.

method called LoftQ for computing a better initialization for quantized training. Additional variations of LoRA have been proposed such as VeRA (Kopiczko et al., 2023) which freezes random weight tied adapters and learns vector scalings of the internal adapter activations. This achieves a further reduction in the number of trainable parameters while achieving comparable performance to LoRA on several NLP finetuning tasks. However, to the best of our knowledge, there is no principled guidance for setting LoRA learning rate which is the focus of our work.

Contributions. We provide guidelines for setting the learning rate through a theory of scaling for neural networks. There is a significant number of works on the scaling of neural networks from the infinite width/depth perspective. The approach is simple: take the width/depth of a neural network to infinity,[1] understand how the limit depends on the choice of the hyperparameters in the training process such as the learning rate and initialization variance, then derive principled choices for these hyperparameters to achieve some desired goal (e.g. improve feature learning). Examples of the infinite-width limit include works on initialization schemes such as (He et al., 2016; Yang, 2019), or more holistically network parametrizations such as (Yang and Hu, 2021) where the authors introduced μP, a neural network parameterization ensuring feature learning in the infinite-width limit, offering precise scaling rules for architecture and learning rates to maximize feature learning. Examples for the depth limit include initialization strategies (Schoenholz et al., 2017a; He et al., 2023; Hayou et al., 2019), block scaling (see e.g. (Hayou et al., 2021; Hayou, 2023; Noci et al., 2023)), depth parametrizations (Yang et al., 2023; Bordelon et al., 2023) etc. Here we propose to use the same strategy to derive scaling rules for the learning rate in LoRA for finetuning. More precisely, we study the infinite-width limit of LoRA finetuning dynamics and show that standard LoRA setup is suboptimal. We correct this by introducing a new method called LoRA+ that improves feature learning in low rank adaptation in the this limit. The key innovation in LoRA+ is setting different learning rates for A and B modules (LoRA modules) as explained in Figure 1. Our theory is validated with extensive empirical results with different language of models and tasks.

2. Setup and Definitions

Our methodology in this paper is model agnostic and applies to general neural network models. Let us consider a neural network of the form

$$\begin{cases} Y_{in}(x) = W_{in}x, \\ Y_l(x) = \mathcal{F}_l(W_l, Y_{l-1}(x)),\ l \in [L], \\ Y_{out}(x) = W_{out}Y_L(x), \end{cases} \tag{1}$$

where $x \in \mathbb{R}^d$ is the input, $L \geq 1$ is the network depth, $(\mathcal{F}_l)_{l\in[L]}$ are mappings that define the layers, $W_l \in \mathbb{R}^{n\times n}$ are the hidden weights, where n is the network *width*, and W_{in}, W_{out} are input and output embedding weights.

Model (1) is *pretrained* on some dataset $\mathcal{D}$ to perform some specified task (e.g. next token prediction). Once the model is pretrained, one can finetune it to improve performance on some downstream task. To achieve this with relatively small devices (limited GPUs), resource-efficient finetuning methods like LoRA significantly reduce the computational cost by considering low rank weight matrices instead of full rank finetuning (or simply full finetuning).

Definition 1 (Low Rank Adapters (LoRA) from (Hu et al., 2021))**.** *For any weight matrix $W \in \mathbb{R}^{n_1\times n_2}$ in the pretrained model, we constrain its update in the fine-tuning process by representing the latter with a low-rank decomposition $W = W^* + \frac{\alpha}{r}BA$. Here, only the weight matrices $B \in \mathbb{R}^{n_1\times r}$, $A \in \mathbb{R}^{r\times n_2}$ are trainable. The rank $r \ll \min(n_1, n_2)$ and $\alpha \in \mathbb{R}$ are tunable constants.*

Scaling of Neural Networks. It is well known that as the width n grows, the network initialization scheme and the learning should be adapted to avoid numerical instabilities and ensure efficient learning. For instance, the variance of the initialization weights (in hidden layers) should scale $1/n$ to prevent arbitrarily large pre-activations as we increase model width n (e.g. He init (He et al., 2016)). To derive such scaling rules, a principled approach consist

[1] Depending on the model, one might want to scale width with fixed depth and vice-versa, or both at the same time. See Appendix A.1 for more details.

of analyzing statistical properties of key quantities in the model (e.g. pre-activations) as n grows and then adjust the initialization, the learning rate, and the architecture itself to achieve desirable properties in the limit $n \to \infty$ (Hayou et al., 2019; Schoenholz et al., 2017b; Yang, 2019; Yang and Littwin, 2023). This approach is used in this paper to study feature learning dynamics with LoRA in the infinite-width limit. This will allow us to derive scaling rules for the learning rates of LoRA modules. For more details about the theory of scaling of neural networks, see Appendix A.1.

Notation. Hereafter, we use the following notation to describe the asymptotic behaviour as the width n grows. Given sequences $c_n \in \mathbb{R}$ and $d_n \in \mathbb{R}^+$, we write $c_n = \mathcal{O}(d_n)$, resp. $c_n = \Omega(d_n)$, to refer to $c_n < \kappa d_n$, resp. $c_n > \kappa d_n$, for some constant $\kappa > 0$. We write $c_n = \Theta(d_n)$ if both $c_n = \mathcal{O}(d_n)$ and $c_n = \Omega(d_n)$ are satisfied. For vector sequences $c_n = (c_n^i)_{1 \leq i \leq k} \in \mathbb{R}^k$ (for some $k > 0$), we write $c_n = \mathcal{O}(d_n)$ when $c_n^i = \mathcal{O}(d_n^i)$ for all $i \in [k]$, and same holds for other asymptotic notations. Finally, when the sequence c_n is a vector of random variables, convergence is understood to be convergence in second moment (L_2 norm).

3. An Intuitive Analysis of LoRA

Our intuition is simple: the matrices A and B have "transposed" shapes and one would naturally ask whether the learning rate should be set differently for the two matrices. In practice, most SOTA models have large width (embedding dimension). Thus, it makes sense to study the training dynamics when the width goes to infinity.

3.1. LoRA with a Toy Model

Consider the following linear model

$$f(x) = (W^* + ba^\top)x, \tag{2}$$

where $W^* \in \mathbb{R}^{1\times n}$ are the pretrained weights, $b \in \mathbb{R}, a \in \mathbb{R}^n$ are LoRA weights,[2] $x \in \mathbb{R}^n$ is the model input. This setup corresponds to $n_1 = 1, n_2 = n, r = 1$ in Definition 1. We assume that the weights W^* are fixed (from pretraining). The goal is to minimize the loss $\mathcal{L}(\theta) = \frac{1}{2}(f(x)-y)^2$ where $\theta = (a, b)$ and (x, y) is an input-output datapoint.[3] We assume that $x = \Theta_n(1)$ which means that input coordinates remain of the same order as we increase width. In the following, we analyze the behaviour of the finetuning dynamics as model width n grows.

Initialization. We consider a Gaussian initialization of the weights as follows: $a_i \sim \mathcal{N}(0, \sigma_a^2)$, $b \sim \mathcal{N}(0, \sigma_b^2)$.[4] With LoRA, we generally want to initialize the product $ba^\top$ to be 0 so that finetuning starts from the pretrained model. This implies at least one of the weights a and b is initialized to 0. If both are initialized to 0, it is trivial that no learning occurs in this case since this is a saddle point. Thus, we should initialize one of the parameters a and b to be non-zero and the other to be zero. If we choose a non-zero initialization for a, then following standard initialization schemes (e.g., He Init (He et al., 2016), LeCun Init (LeCun et al., 2002)), one should set $\sigma_a^2 = \Theta(n^{-1})$ to ensure $a^\top x$ does not explode with width. This is justified by the Central Limit Theorem (CLT).[5] On the other hand, if we choose a non-zero initialization for b, one should make sure that $\sigma_b^2 = \Theta(1)$. This leaves us with two possible schemes:

- `Init[1]`: $\sigma_b^2 = 0, \sigma_a^2 = \Theta(n^{-1})$.
- `Init[2]`: $\sigma_b^2 = \Theta(1), \sigma_a^2 = 0$.

Our analysis will only consider these two initialization schemes for LoRA modules, although the results should in-principle hold for other schemes, providing that stability (as discussed above) is satisfied.

Learning rate. WLOG, we can simplify the analysis by assuming that $W^* = 0$. This can be achieved by setting $\tilde{y} = y - W^*x$. The gradients are given by

$$\frac{\partial \mathcal{L}}{\partial b} = a^\top x(f(x) - y), \quad \frac{\partial \mathcal{L}}{\partial a} = b(f(x) - y)x.$$

We use subscript t to denote the finetuning step. Let $U_t = (f_t(x) - y)$. At step t with learning rate $\eta > 0$, we have

$$\Delta f_t \overset{def}{=} f_t(x) - f_{t-1}(x) = -\underbrace{\eta b_{t-1}^2 U_{t-1}\|x\|^2}_{\delta_t^1}$$
$$- \underbrace{\eta(a_{t-1}^\top x)^2 U_{t-1}}_{\delta_t^2} + \underbrace{\eta^2 U_{t-1}^2 b_{t-1}(a_{t-1}^\top x)\|x\|^2}_{\delta_t^3}.$$

The update in model output is driven by the three terms $(\delta_t^i)_{i\in\{1,2,3\}}$. The first two terms represent "linear" contributions to the update, i.e. change in model output driven by fixing b and updating a and vice-versa. These terms are order one in η. The third term δ_t^3 represents a multiplicative update, compounding the updates in a and b, and is an order two term in η. As n grows, a desirable property is that $\Delta f_t = \Theta(1)$. Intuitively, this means

[2] Here, we consider $n_2 = 1$ to simplify the analysis. All the conclusions remain essentially valid when $n_2 = n_1 = n$.

[3] For simplicity, we assume that the finetuning dataset consists of a single sample. Our analysis is readily generalizable to multiple samples.

[4] The Gaussian distribution can be replaced by any other distribution with finite variance.

[5] Technically, the CLT only ensures the almost sure convergence, the L_2 convergence follows from the Dominated Convergence Theorem. We omit these technical details in this paper.

that as we scale the width, feature updates do not ‘suffer’ from this scaling (see Appendix A.1 for more details). An example of a scenario where feature learning is affected by scaling is the lazy training regime (Jacot et al., 2018), where feature updates are of order $\Theta(n^{-1/2})$ which implies that no feature learning occurs in the limit $n \to \infty$. The condition $\Delta f_t = \Theta(1)$ also implies that the update does not explode with width, which is also a desirable property.

Having $\Delta f_t = \Theta(1)$ satisfied implies that at least one of the three terms $(\delta_t^i)_{i\in\{1,2,3\}}$ is $\Theta(1)$. Ideally, we want both δ_t^1 and δ_t^2 to be $\Theta(1)$ because otherwise it means that either a or b is not efficiently updated. For instance, if $\delta_t^1 = o(1)$, it means that as $n \to \infty$, the model acts as if a is fixed and only b is trained. Similar conclusions hold when $\delta_t^2 = o(1)$. Having both δ_t^1 and δ_t^2 being $\Theta(1)$ in width means that both a and b parameter updates significantly contribute to the change in $f_t(x)$, and we say that feature learning with LoRA is *efficient* when this is the case, i.e. $\delta_i^t = \Theta(1)$ for $i \in \{1,2\}$ and all $t > 1$. We will formalize this definition of efficiency in the next section. The reader might wonder why we do not require that δ_t^3 be $\Theta(1)$. We will see that when both δ_t^1 and δ_t^2 are $\Theta(1)$, the term δ_t^3 is also $\Theta(1)$.

Efficiency Analysis. Let us assume that we train the model with gradient descent with learning rate $\eta = \Theta(n^c)$ for some $c \in \mathbb{R}$, and suppose that we initialize the model with `Init[1]`. Sine the training dynamics are mainly matrix vector products, sum of vectors/scalars etc (see (Yang et al., 2022)),[6] it is easy to see that any quantity in the training dynamics should be of order n^γ for some $\gamma \in \mathbb{R}$. For any quantity v in the training dynamics, we write $v = \Theta(n^{\gamma[v]})$. When v is a vector, we use the same notation when all entries of v are $\Theta(n^{\gamma[v]})$. The γ notation is formally defined in Appendix A.

Starting from initialization, we have $f_0(x) = 0$. LoRA finetuning is efficient when $\delta_t^1 = \Theta(1)$ and $\delta_t^2 = \Theta(1)$ for all $t > 1$,[7] and $f_t(x) = \Theta(1)$ for $t > 1$. This translate to

$$\begin{cases} c + 2\gamma[b_{t-1}] + 1 = 0 \quad (\delta_t^1 = \Theta(1)) \\ c + 2\gamma[a_{t-1}^\top x] = 0 \quad (\delta_t^2 = \Theta(1)) \\ \gamma[b_{t-1}] + \gamma[a_{t-1}^\top x] = 0 \quad (f_{t-1}(x) = \Theta(1)) \end{cases}$$

Solving this equation yields $c = -1/2$, i.e. the learning rate should scale as $\eta = \Theta(n^{-1/2})$ in order to achieve efficient feature learning. At initialization, $b_0 = 0$ and $a_0^\top x = \Theta(1)$ (by Central Limit Theorem). Through an inductive argument, for $t > 0$, b_t will be of order $\Theta(n^{-1/2})$ and $a_t^\top x$ will be of order $\Theta(1)$, yielding $f_t(x) = \Theta(n^{-1/2})$. Indeed, at each iteration the update to b_t will be of order $\Theta(\eta y a_{t-1}^\top x) = \Theta(n^{-1/2})$ and the updates to a_t are of order $\Theta(\eta b_{t-1} y x) = \Theta(n^{-1})$. As $f_t = \Theta(n^{-1/2})$, this yields a contradiction towards learning $\Theta(1)$ features.

This shows that we cannot have both δ_t^1 and δ_t^2 to be $\Theta(1)$ with this parametrization (also true with `Init[2]`). We formalize this result in the next proposition and refer the reader to Appendix A for further technical details.

Proposition 1 (Inefficiency of LoRA fine-tuning). *Assume that LoRA weights are initialized with `Init[1]` or `Init[2]` and trained with gradient descent with learning rate $\eta = \Theta(n^c)$ for some $c \in \mathbb{R}$. Then, it is impossible to have $\delta_t^i = \Theta(1)$ for $i \in \{1,2\}$ for any $t > 0$, and therefore, fine-tuning with LoRA in this setup is inefficient.*

In conclusion, efficiency cannot be achieved with this parametrization of the learning rate. This suggests that standard LoRA finetuning as currently used by practitioners is suboptimal, especially when model width is large, which is a property that is largely satsified in practice ($n \approx 700$ for GPT2 and $n \approx 4000$ for LLama). This analysis suggests that *we are missing crucial hyperparameters* in the standard LoRA setup. Indeed, we show that by decoupling the learning rate for a and b, we can have $\delta_t^i = \Theta(1)$ for $i \in \{1,2,3\}$. We write η_a, η_b to denote the learning rates. The analysis conducted above remains morally the same with the only difference being in the learning rates. Let $\eta_a = \Theta(n^{c_a})$ and $\eta_b = \Theta(n^{c_b})$, and assume that weights are initialized with `Init[1]`. A similar analysis to the one conducted above show that having $f_t(x) = \Theta(1)$ and $\delta_t^i = \Theta(1)$ for $i \in \{1,2\}$ and $t > 0$ implies that for all $t > 1$

$$\begin{cases} c_a + 2\gamma[b_{t-1}] + 1 = 0 \quad (\delta_t^1 = \Theta(1)) \\ c_b + 2\gamma[a_{t-1}^\top x] = 0 \quad (\delta_t^2 = \Theta(1)) \\ \gamma[b_{t-1}] + \gamma[a_{t-1}^\top x] = 0 \quad (f_{t-1}(x) = \Theta(1)) \end{cases}$$

which, after simple calculations, implies that $c_a + c_b = -1$. This is only a necessary condition. In the next result, taking also some elements of stability into consideration, we fully characterize the choice of η_a and η_b to ensure efficient LoRA fine-tuning.

Proposition 2 (Efficient Fine-Tuning with LoRA). *In the case of model (2), with $\eta_a = \Theta(n^{-1})$ and $\eta_b = \Theta(1)$, we have for all $t > 1$, $i \in \{1,2,3\}$, $\delta_t^i = \Theta(1)$.*

We refer the reader to Appendix A for more details on the proof of Proposition 2. In conclusion, scaling the learning

[6] A crucial assumption for this to hold is also to have that for any matrix/vector product in the training dynamics, the product dimension (the dimension along which the matrix/vector product is calculated) is $\Theta(n^\alpha)$ for some $\alpha > 0$. For instance, in the case of Transformers, this is satisfied since the MLP embedding dimension is generally $k \times n$. However, this condition would be violated if for instance one considers MLP embedding dimension $kn\log(n)$. Such non-standard scaling choices require a particular treatment, but the conclusions remain the same.

[7] Here we use the $t > 1$ instead of $t > 0$ because at $t \leq 1$, at least one the terms δ_1^1 or δ_1^2 will be zero.

rates as $\eta_a = \Theta(n^{-1})$ and $\eta_b = \Theta(1)$ ensures stability ($\Delta f_t = \Theta(1)$) and efficiency of LoRA finetuning ($\delta_t^i = \Theta(1)$ for $i \in \{1, 2\}$ and $t > 1$) in the infinite-width limit. In practice, this means that the learning rate for b should be generally much larger than that of a. This remains true even if $b \in \mathbb{R}^r$ for general r. We will later see that this scaling is valid for general neural network models.

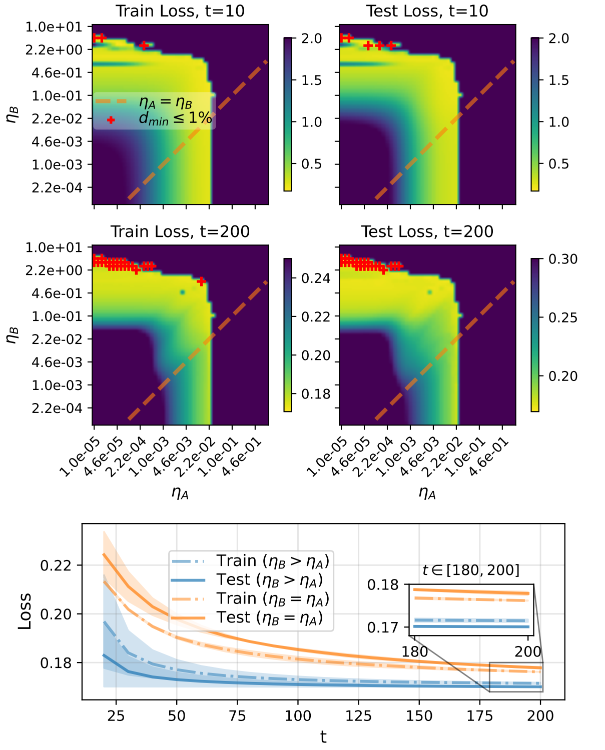


Figure 2. (**Top**) Train/Test accuracy of toy model Equation (3) averaged over 3 random seeds. Orange dashed line represents the line $\eta_A = \eta_B$, and red dots represents all values of (η_A, η_B) for which $d_{\min}(\eta_A, \eta_B) := \mathcal{L}_{(\eta_A, \eta_B)}/\mathcal{L}^* - 1 \leq 1\%$, where $\mathcal{L}^*$ is the best loss. (**Bottom**) Train/Test curves for two sets of learning rates: the optimal choice $(\eta_A^*, \eta_B^*) = (2.78, 1.29\text{e}{-4})$ overall at $t = 200$ in terms of test loss (Blue) and the optimal choice when $\eta_A = \eta_B$ which is given by $(\eta_A, \eta_B) = (2.15\text{e}{-2}, 2.15\text{e}{-2})$ (Orange). All values are averaged oevr three runs and confidence interval are shown (shaded).

3.2. Verifying the Results on a Toy Model

The previous analysis considers a simple linear model. To assess the validity of the scaling rules in a non-linear setting, we consider a neural network model given by

$$f(x) = W_{out}\phi(BA\phi(W_{in}x)), \tag{3}$$

where $W_{in} \in \mathbb{R}^{n\times d}, W_{out} \in \mathbb{R}^{1\times n}, A \in \mathbb{R}^{r\times n}, B \in \mathbb{R}^{n\times r}$ are the weights, and ϕ is the ReLU function. The model is trained on a synthetic dataset generated with $X \sim \mathcal{N}(0, I_d)$, $Y = \sin(d^{-1}\sum_{i=1}^d X_i)$. See Appendix C for more details.

Only the weight matrices A, B are trained (W_{in}, W_{out} are fixed). We use $d = 5, n = 100, r = 4$, train data size 1000 and a test data size 100.[8] The train/test loss for varying η_A and η_B is reported in Figure 2 at the early stages of the training ($t = 10$) and after convergence (we observed convergence around $t \approx 200$ for reasonable choices of learning rates). The red '+' signs represents learning rates (η_A, η_B) for which the loss is within 1% range from the best loss and dashed line represents the case where the learning rates are set equal. We observe that both the best train and test losses are consistently achieved by a combination of learning rates where $\eta_b \gg \eta_a$, which validates our analysis in the previous section. Notice also that optimal learning rates (η_A, η_B) are generally close to the edge of stability, a well-known behaviour in training dynamics of deep networks (Cohen et al., 2021).

4. Stability and Feature Learning with LoRA in the Infinite Width Limit

In this section, we extend the analysis above to general neural architectures with LoRA layers. We show that the conclusions from the analysis on the linear model hold for general neural architectures: 1) using the same learning rate for both A and B leads to suboptimal feature learning when model width is large, and 2) this problem can be fixed by setting different learning rates for A and B.

Since our aim in this paper is primarily methodological, the theoretical results in this section are of a physics level of rigor, omitting technical assumptions that would otherwise make the analysis rigorous but unnecessarily complicated. In all the results, LoRA rank r is considered fixed and finetuning dynamics are analyzed in the limit of infinite-width. This setup fairly represents practical scenarios where $r \ll n$ and r is generally small.

Notation. The LoRA weights are initialized with $A_{ij} \sim \mathcal{N}(0, \sigma_A^2), B_{ij} \sim \mathcal{N}(0, \sigma_B^2)$ for some $\sigma_A, \sigma_B \geq 0$.[9] Here also, we assume that either $\sigma_B^2 = 0$ and $\sigma_A^2 = \Theta(n^{-1})$ (`Init[1]`), or $\sigma_B^2 = \Theta(1)$ and $\sigma_A^2 = 0$ (`Init[2]`). Given a LoRA layer in the model, $\underline{Z}$ denotes the input to that layer and $\bar{Z}$ the output after adding the pretrained weights. More precisely, we write $\bar{Z} = W^*\underline{Z} + \frac{\alpha}{r}BA\underline{Z}$.

Our main analysis relies on a careful estimation of the magnitude of several quantities including *LoRA features*. Let us first give a formal definition.

Definition 2 (LoRA Features)**.** *Given a general neural architecture and a LoRA layer (Definition 1), we define LoRA features* (Z_A, Z_B) *as* $Z_A = A\underline{Z}$ *and* $Z_B = BZ_A =$

[8] See Appendix C for more details about the experimental setup.

[9] In (Hu et al., 2021), B is initialized to 0, which corresponds to setting $\sigma_B = 0$.

$BA\underline{Z}$. At fine-tuning step t, we use the superscript t to denote the value of LoRA features Z_A^t, Z_B^t, and the subscript t to denote the weights A_t, B_t.

LoRA layers are 2-layers linear networks with a "bottleneck" in the middle (since generally $r \ll n$). This bottleneck shape might induce some numerical challenges in training stability and efficiency (Definition 3 and Definition 5).

Finetuning Dataset. To simplify the analysis, we assume that the finetuning dataset comprises a single sample (x, y),[10] and the goal is to minimize the loss $\mathcal{L}(\boldsymbol{\theta}, (x, y))$ computed with the underlying model where the adjusted weights are given by $W^* + BA$ for all LoRA layers (here $\boldsymbol{\theta} = \{A, B,$ for all LoRA layers in the model$\}$). At training step t, and for any LoRA layer in the model, $\underline{Z}^t$ is the input to the LoRA layer, computed with data input x. Similarly, we write $d\bar{Z}^t$ to denote the gradient of the loss function with respect to the layer output features $\bar{Z}$ evaluated at data point (x, y).

The notion of stability of LoRA as discussed in Section 3 can be generalized to any neural network model as follows.

Definition 3 (Stability)**.** *We say that LoRA finetuning is stable if for all LoRA layers in the model, and all training steps t, we have $\underline{Z}, Z_A, Z_B = \mathcal{O}(1)$ as n goes to infinity.*

Stability implies that no quantity in the network explodes as width grows, a desirable property as we scale the model.[11] Naturally, in order to ensure stability, one has to scale hyperparameters (initialization, learning rate) as n grows. Scaling rules for initialization are fairly easy to infer and were already discussed in Section 3 where we obtained two plausible initialization schemes (`Init[1]` and `Init[2]`). More importantly, if we arbitrarily scale the learning rate with width, we might end up with suboptimal learning as width grows even if the finetuning is stable. This is the case for instance when we aggressively downscale the learning rate with width, or inadequately parameterize the network (e.g. Neural Tangent Kernel parametrization which leads to the kernel regime in the infinite width limit, (Jacot et al., 2018)). To take this into account, we define a notion of feature learning with LoRA.

Definition 4 (Stable Feature Learning with LoRA)**.** *We say that LoRA finetuning induces stable feature learning if it is stable (Definition 3), and for all LoRA layers and finetuning step t, we have $\Delta Z_B^t \overset{def}{=} Z_B^{t+1} - Z_B^t = \Theta(1)$.*

A similar definition of feature learning was introduced in (Yang and Littwin, 2023) for pretraining. This definition ensures that the network is not 'stuck' in a kernel regime where feature updates are of order $\mathcal{O}(n^{-\epsilon})$ in the infinite-width limit for some $\epsilon > 0$, which implies that no feature learning occurs in the limit. The authors introduced the μ-parameterization (or maximal update parametrization), a specific network parameterization (initialization + learning rate scaling), that ensures that feature updates are $\Theta(1)$. Note that here we added stability in the definition, but in principle, one could define feature learning with Ω instead of Θ. The latter covers unstable scenarios (e.g. when $\Delta Z_B^t = \Theta(n)$ due to improper scaling of initialization and learning rate), so we omit it here and focus on stable feature learning. Also, notice that we only consider finetuning dynamics and not the pretraining dynamics. However, since our analysis depends on weights W^* from pretraining, we assume that pretraining parameterization ensures stability and feature learning as width grows (see Appendix A for more details).[12]

At finetuning step t, the gradients are given by

$$\frac{\partial \mathcal{L}_t}{\partial B} = \frac{\alpha}{r} d\bar{Z}^{t-1} \otimes A_{t-1}\underline{Z}^{t-1}$$
$$\frac{\partial \mathcal{L}_t}{\partial A} = dZ_A^{t-1} \otimes \underline{Z}^{t-1} = \frac{\alpha}{r} B_{t-1}^\top d\bar{Z}^{t-1} \otimes \underline{Z}^{t-1},$$

where $u \otimes v$ denotes the outer product $uv^\top$ of vectors u, v, and the weights are updated as follows

$$A_t = A_{t-1} - \eta_A g_A^{t-1}, \quad B_t = B_{t-1} - \eta_B g_B^{t-1},$$

where g_A, g_B are processed gradients (e.g. normalized gradients with momentum as in AdamW etc). Hereafter, we assume that the gradients are processed in a way that makes their entries $\Theta(1)$. This is generally satisfied in practice (with Adam for instance) and has been considered in (Yang and Littwin, 2023) to derive the μ-parametrization for general gradient processing functions.

Unlike the linear model in Section 3, LoRA feature updates are not only driven by the change in the A, B weights, but also $\underline{Z}, d\bar{Z}$ which are updated as we finetune the model (assuming there are multiple LoRA layers). To isolate the contribution of individual LoRA layers to feature learning,

[10] This assumption on the finetuning dataset is for simplification purposes only. All our analysis can be re-written with 'batched' gradients and the conclusions remain the same. However, some additonal assumptions are required to make the analysis rigorous.

[11] It is possible to define stability as $\underline{Z}, Z_B = \mathcal{O}(1)$ and exclude Z_A from the condition. This would allow scenarios where for instance the entries of A explode with width but their magnitude is compensated with a smaller magnitude of B. This system has one degree of freedom because of the homogeneity of the product BA, and by imposing that $Z_A = \mathcal{O}(1)$, we avoid having such scenarios.

[12] When taking the infinite width limit, we assume that pretraining parameterization is μP. This is just a technicality for the infinite-width limit and does not have any implications on practical scenarios where the width is finite. The most important implications of this assumption is that in the pretrained network (before introducing LoRA layers), we have $\underline{Z} = \Theta(1), \bar{Z} = \Theta(1)$, which holds for a general input-output pair (x, y).

we assume that only a *single LoRA layer is trainable* and all other LoRA layers are frozen.[13]. In this setting, considering the only trainable LoRA layer in the model, the layer input $\underline{Z}$ is fixed and does not change with t, while $d\bar{Z}$ changes with step t (because $\bar{Z}^t = (W^* + \frac{\alpha}{r}B_t A_t)\underline{Z}$). After step t, Z_B is updated as follows

$$\Delta Z_B^t = \underbrace{B_{t-1}\Delta Z_A^t}_{\delta_t^1} + \underbrace{\Delta B_t Z_A^{t-1}}_{\delta_t^2} + \underbrace{\Delta B_t \Delta Z_A^t}_{\delta_t^3}$$

As discussed in Section 3, the terms δ_t^1, δ_t^2 represent the 'linear' feature updates that we obtain if we fix one weight matrix and only train the other, while δ_t^3 represents the 'multiplicative' feature update which captures the compounded update due to updating both A and B.

Analysis of the Role of A and B. As discussed above, we want to ensure that $\delta_t^1 = \Theta(1)$ and $\delta_t^2 = \Theta(1)$ which means that both weight matrices contribute to the update in Z_B. To further explain why this is a desirable property, let us analyze how changes in matrices A and B affect LoRA feature $Z_B = BA\underline{Z}$.

Let $(B_{:,i})_{1\leq i\leq r}$ denote the columns of B. We can express Z_B as $Z_B = \sum_{i=1}^r (A\underline{Z})_i B_{:,i}$, where $(A\underline{Z})_i$ is the i^{th} coordinate of $A\underline{Z}$. This decomposition suggests that the *direction* of Z_B is a weighted sum of the columns of B, and A modulates the *weights*. With this, we can also write

$$\begin{cases} \delta_t^1 = \sum_{i=1}^r (\Delta A_t \underline{Z})_i (B_{:,i})_{t-1} \\ \delta_t^2 = \sum_{i=1}^r (A_{t-1}\underline{Z})_i (\Delta B_{:,i})_{t-1}, \end{cases}$$

where $(B_{:,i})_t$ refers to the columns of B at time step t. Having both δ_t^1 and δ_t^2 of order $\Theta(1)$ means that both A and B are 'sufficiently' updated to induce a change in weights $(A\underline{Z})_i$ and directions $B_{:,i}$. If one of the matrices A, B is not efficiently updated, we might end up with suboptimal finetuning, leading to either non updated directions B or direction weights $(A_{t-1}Z)$. For instance, assuming that the model is initialized with `Init[2]`, and that B is not efficiently updated, the direction of Z_B will be mostly determined by the vector (sub)space of dimension r generated by the columns of B at initialization. This analysis leads to the following definition of efficient learning with LoRA.

Definition 5 (Efficient Learning)**.** *We say that LoRA finetuning is efficient if it is stable (Definition 3), and for all LoRA layers in the model, all steps $t > 1$, and $i\{1, 2\}$, we have $\delta_t^i = \Theta(1)$.*

Note that it is possible to achieve stable feature learning (Definition 4) without necessarily having efficient learning. This is the case when for instance B is not updated (fixed to a non-zero init with `Init[2]`) and only A is updated, which corresponds to simply setting $\eta_B = 0$. This is a trivial case, but other non-trivial cases of inefficiency are common in practice, such as the use of the same learning rate for A and B which is a standard practice. In the next theorem, we characterize the optimal scaling of learning rates η_A and η_B, a conclusion similar to that of Section 3.

Theorem 1 (Efficient LoRA (Informal))**.** *Assume that weight matrices A and B are trained with Adam with respective learning rates η_A and η_B. Then, it is impossible to achieve efficiency with $\eta_A = \eta_B$. However, LoRA Finetuning is efficient with $\eta_A = \Theta(n^{-1})$ and $\eta_B = \Theta(1)$.*

The result of Theorem 1 suggests that efficiency can only be achieved with $\eta_B/\eta_A = \Theta(n)$. In practice, this translates to setting $\eta_B \gg \eta_A$, but does not provide a precise ratio η_B/η_A to be fixed while tuning the learning rate (the constant in 'Θ' is generally intractable), unless we tune both η_B and η_A which is not efficient from a computational perspective as it becomes a 2D tuning problem. It is therefore natural to set a fixed ratio η_B/η_A and tune only η_A (or η_B), which would effectively reduce the tuning process to a 1D grid search, achieving the same computational cost of standard LoRA where the learning rate is the same for A and B. We call this method LoRA+.

LoRA+ : set the learning rates for A, B such that $\eta_B = \lambda\eta_A$ with $\lambda > 1$ fixed and tune η_A.

In the next section, through extensive empirical evaluations, we first validate our theoretical result and show that optimal pairs (η_A, η_B) (in terms of test accuracy) generally satisfy $\eta_B \gg \eta_A$. We then investigate the optimal ratio λ for LoRA+ and suggest a default ratio that was empirically found to generally improve performance compared to standard LoRA. Although the conclusions of Theorem 1 and Proposition 2 are similar, the proof techniques are different. In Proposition 2, the linear model is trained with gradient descent, while in Theorem 1, the training algorithm is Adam-type in the sense that it normalizes the gradients before updating the weights. The formal statement of Theorem 1 requires an additional assumption on the alignment of the processed gradients g_A with LoRA input $\underline{Z}$. This technical detail is introduced and discussed in Appendix A.

5. Experiments with Language Models

We report our empirical results using LoRA to finetune a set of language models on different benchmarks. Details about the experimental setup and more empirical results are provided in Appendix C. We also identify a default value for the ratio $\lambda = \eta_B/\eta_A$ that generally improves

[13] This is equivalent to having only a single LoRA layer in the model since LoRA layers are initialized to zero. In this way, we can quantify feature learning induced by the LoRA layer as we finetune the model.

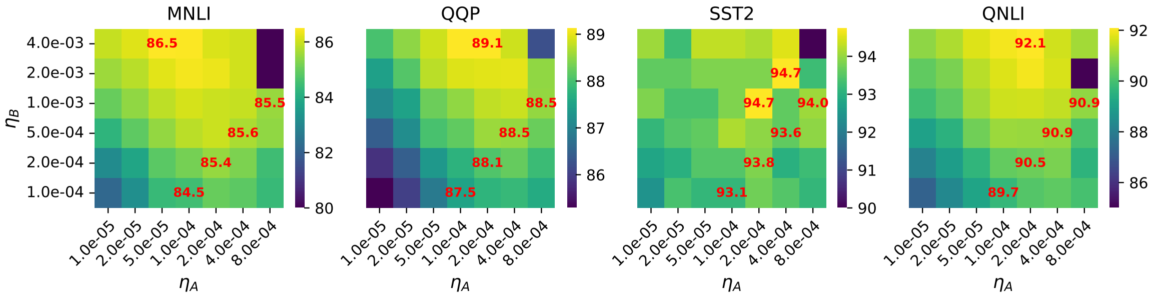


Figure 3. Test accuracy of Roberta-base finetuning for 3 epochs on MNLI, QQP, QNLI, and 10 epochs on SST2, with sequence length $T = 128$ and half precision (FP16). LoRA hyperparameters are set to $\alpha = r = 8$. All values are averaged over 3 random seeds (we do not show confidence intervals for better visualizations, but fluctuations are of order 0.1%, see Figure 7 for instance). For better visualization, when accuracy is lower than a fixed threshold, we set it to threshold. Values shown in red are: 1) the best accuracy (overall) and 2) the accuracy for a set of learning rates where η_B and η_A are close in order of magnitude ($\eta_B/\eta_A \in [1, 1.25]$).

performance as compared to standard LoRA. The code for our experiments is available at https://github.com/nikhil-ghosh-berkeley/loraplus.

5.1. GLUE tasks with GPT-2 and RoBERTa

The GLUE benchmark (General Language Understanding Evaluation) consists of several language tasks that evaluate the understanding capabilities of langugage models (Wang et al., 2018). Using LoRA, we finetune Roberta-base from the RoBERTa family (Liu et al., 2019) and GPT-2 (Radford et al., 2019) on MNLI, QQP, SST2, and QNLI tasks (Other tasks are smaller and generally require an already finetuned model e.g. on MNLI as starting checkpoint) with varying learning rates (η_A, η_B) to identify the optimal combination. Empirical details are provided in Appendix C.

Roberta-base. Figure 3 shows the results of Roberta-base finetuning with $\alpha = r = 8$, trained with half precision (FP16). We observe that test accuracy is consistently maximal for some set of learning rates satisfying $\eta_B \gg \eta_A$, outperforming the standard practice where η_A and η_B are usually set equal. Interestingly, the gap between the optimal choice of learning rates overall and the optimal choice when $\eta_A \approx \eta_B$ is more pronounced for 'harder' tasks like MNLI and QQP, as compared to SST2 and QNLI. This is probably due to the fact that harder tasks require more efficient feature learning. It is also worth mentioning that in our experiments, given limited computational resources, we use sequence length $T = 128$ and finetune for only 3 epochs for MNLI and QQP, so it is expected that we obtain test accuracies lower that those reported in (Hu et al., 2021) where the authores finetune Roberta-base with $T = 512$ sequence length (for MNLI) and more epochs (30 for MNLI). In Appendix C, we provide additional results with Test/Train accuracy/loss.

GPT-2. Figure 4 shows the results of finetuning GPT-2 with LoRA on MNLI and QQP (other tasks and full precision training are provided in Appendix C). Similar to the conclusions from Roberta-base, we observe that maximal test accuracies are achieved with some (η_A, η_B) satisfying $\eta_B \gg \eta_A$. Further GPT-2 results with different tasks are provided in Appendix C. Here also, we observed that the harder the task, the larger the gap between model performance when $\eta_B \gg \eta_A$ and when $\eta_A \approx \eta_B$.

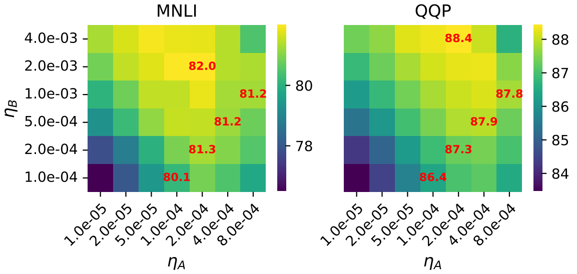


Figure 4. Test accuracy of GPT-2 after finetuning for 3 epochs on MNLI, QQP, with FP16 precision. LoRA hyperparameters are set to $\alpha = r = 8$. Both train/test accuracy are consistently maximal for some choice of learning rates where $\eta_B \gg \eta_A$. See Appendix C for more numerical results with GPT2.

5.2. Llama

To further validate our theoretical findings, we finetune the Llama-7b model (Touvron et al., 2023) on the MNLI dataset and flan-v2 dataset (Longpre et al., 2023) using LoRA. Each trial is averaged over two seeds.

Flan-v2. We examine LoRA training of Llama on the instruction finetuning dataset flan-v2 (Longpre et al., 2023). To make the experiments computationally feasible, we train for one epoch on a size $100,000$ subset of the flan-v2 dataset. We record the test accuracy of the best checkpoint every 500 steps. The LoRA hyperparameters are set to $\alpha = 16$ and $r = 64$. The adapters are added to every linear layer (excluding embedding layers) and we use a constant learning rate schedule. The full training details are in Appendix C.

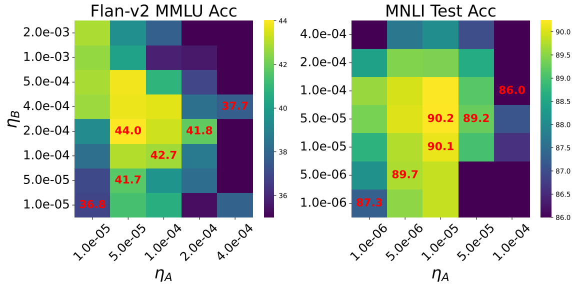


Figure 5. Left: MMLU accuracy of Llama-7b trained for one epoch on a 100k subset of flan-v2. Right: Test accuracy of the best checkpoint of Llama-7b trained on MNLI for one epoch. Values are averaged over two seeds.

We evaluate the final model on the MMLU benchmark (Hendrycks et al., 2020). The results in Figure 5 show that for this benchmark taking $\eta_B \gg \eta_A$ is advantageous and results in a roughly 1.3% gain compared with the optimal $\eta_B = \eta_A$. In Appendix C we show that the same effect holds also when using `Init[1]`.

MNLI. The right panel of Fig 5 shows the results of finetuning Llama-7b with LoRA on MNLI, with $\alpha = 16$, $r = 8$. We train using half precision and constant learning rate schedule, with a sequence length $T = 128$. Since MNLI is relatively easy for Llama, we finetune for only one epoch, which is sufficient for the model to reach its peak test accuracy. In Figure 5, $\eta_B = \eta_A$ is nearly optimal for all $\eta_B \geq \eta_A$. This is consistent with the intuition that efficient feature learning is not required for easy tasks and that having $\eta_B/\eta_A \gg 1$ does not significantly enhance performance. Additionally, the magnitude of stable learning rates for Llama is much smaller than for GPT-2 and RoBERTa on MNLI further supporting that Llama requires less adaptation. Analogous plots for the train and test loss are shown in Fig 19 in Appendix C.

5.3. How to set LoRA+ Ratio?

Naturally, the optimal ratio λ depends on the architecture and the finetuning task via the constants in 'Θ' (Theorem 1). This is a limitation of these asymptotic results since they do not offer any insights on how the constants are affected by the task and the neural architecture. Figure 6 show the distribution of the ratio η_B/η_A for the top 4 runs in terms of test accuracy for different pairs of (model, task). This is the same experimental setup of Figure 3 and Figure 4. The optimal ratio is model and task sensitive and shows significant variance. Our additional experiments in Appendix C show that it is also sensitive to initialization (`Init[1]` vs `Init[2]`). With `Init[2]`, we found that generally setting a ratio of $\lambda = \eta_B/\eta_A \approx 2^4$ improves performance for Roberta (Figure 7). However, with `Init[1]`, we found that the optimal ratio is smaller and is of order 2^2-2^3 (see Appendix C). For LLama experiments, it seems that a ratio of order 2^1-2^2 is optimal..

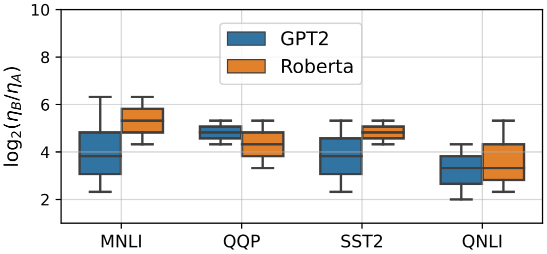


Figure 6. Distribution of the ratio η_B/η_A for the top 4 learning rate for each pair (model, task). The 4 learning rates are selected using the test loss at the end of finetuning (i.e. top 4 learning rates (η_B, η_A) in terms of test loss). The distribution shows the interquartile range ($25\% - 75\%$ quantiles) and the median.

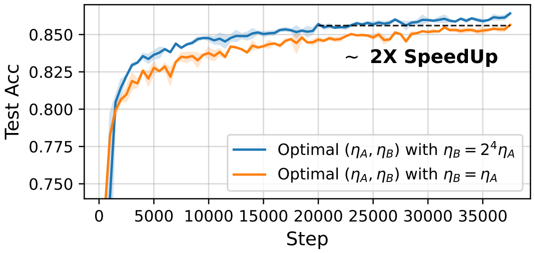


Figure 7. Test accuracy of Roberta-base finetuned on the MNLI task in two setups: (**LoRA+**) $\eta_B = 2^4\eta_A$ and (**Standard**) $\eta_B = \eta_A$. η_A is tuned using a grid search.

6. Conclusion and Limitations

Employing a scaling argument, we showed that LoRA finetuning as it is currently used in practice is not efficient. We proposed a method, LoRA+, that resolves this issue by setting different learning rates for LoRA adapter matrices. Our analysis is supported by extensive empirical results confirming the benefits of LoRA+ for both training speed and performance. These benefits are more significant for 'hard' tasks such as MNLI for Roberta/GPT2 (compared to SST2 for instance) and MMLU for LLama-7b (compared to MNLI for instance). However, as we depicted in Figure 7, a more refined estimation of the optimal ratio η_B/η_A should take into account task and model dependent, and our analysis in this paper lacks this dimension. We leave this for future work.

Acknowledgement

We thank Amazon Web Services (AWS) for cloud credits under an Amazon Research Award. We also gratefully acknowledge partial support from NSF grants DMS-2209975, 2015341, NSF grant 2023505 on Collaborative Research: Foundations of Data Science Institute (FODSI), the NSF and the Simons Foundation for the Collaboration on the Theoretical Foundations of Deep Learning through awards DMS-2031883 and 814639, and NSF grant MC2378 to the Institute for Artificial CyberThreat Intelligence and OperatioN (ACTION).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning, specifically, to speed up the leading algorithm LoRA for fine-tuning pre-trained large language models while improving performance of the fine-tuned models. The speed-up saves computation resources when pre-trained large language models are customized for particular down-stream tasks. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit, 2023.

Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jh-rTtvkGeM.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Soufiane Hayou. On the infinite-depth limit of finite-width neural networks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=RbLsYz1Az9.

Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2672–2680. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hayou19a.html.

Soufiane Hayou, Eugenio Clerico, Bobby He, George Deligiannidis, Arnaud Doucet, and Judith Rousseau. Stable resnet. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 1324–1332. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/hayou21a.html.

Bobby He, James Martens, Guodong Zhang, Aleksandar Botev, Andrew Brock, Samuel L Smith, and Yee Whye Teh. Deep transformers without shortcuts: Modifying self-attention for faithful signal propagation, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*, 2023.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 2023.

Lorenzo Noci, Chuning Li, Mufan Bill Li, Bobby He, Thomas Hofmann, Chris Maddison, and Daniel M. Roy. The shaped transformer: Attention models in the infinite depth-and-width limit, 2023.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9, 2019.

Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation, 2017a.

S.S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein. Deep information propagation. In *International Conference on Learning Representations*, 2017b.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2018.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *arXiv preprint arXiv:2306.04751*, 2023.

G. Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pages 11727–11737. PMLR, 2021.

Greg Yang and Etai Littwin. Tensor programs ivb: Adaptive optimization in the infinite-width limit. *arXiv preprint arXiv:2308.01814*, 2023.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite-depth neural networks. *arXiv preprint arXiv:2310.02244*, 2023.

Liu Yang, Steve Hanneke, and Jaime Carbonell. A theory of transfer learning with applications to active learning. *Machine learning*, 90:161–189, 2013.

Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. *arXiv preprint arXiv:2310.17513*, 2023.

A. Proofs

In this section, we provide proofs for Proposition 1, Proposition 2, Theorem 1, and some technical details used in the proofs.

A.1. Scaling of Neural Networks

Scaling refers to the process of increasing the size of one of the ingredients in the model to improve performance (see e.g. (Hoffmann et al., 2022)). This includes model capacity which can be increased via width (embedding dimension) or depth (number of layers) or both, compute (training data), number of training steps etc. In this paper, we are interested in scaling model capacity via the width n. This is motivated by the fact that most state-of-the-art language and vision models have large width.

It is well known that as the width n grows, the network initialization scheme and the learning should be adapted to avoid numerical instabilities and ensure efficient learning. For instance, the initialization variance should scale $1/n$ to prevent arbitrarily large pre-activations as we increase model width n (e.g. He init (He et al., 2016)). To derive such scaling rules, a principled approach consist of analyzing statistical properties of key quantities in the model (e.g. pre-activations) as n grows and then adjust the initialization, the learning rate, and the architecture itself to achieve desirable properties in the limit $n \to \infty$ (Hayou et al., 2019; Schoenholz et al., 2017b; Yang, 2019).

In this context, (Yang et al., 2022) introduces the Maximal Update Parameterization (or μP), a set of scaling rules for the initialization scheme, the learning rate, and the network architecture that ensure stability and maximal feature learning in the infinite width limit. Stability is defined by $Y_l^i = \Theta(1)$ for all l and i where the asymptotic notation '$\Theta(.)$' is with respect to width n (see next paragraph for a formal definition), and feature learning is defined by $\Delta Y_l = \Theta(1)$, where Δ refers to the feature update after taking a gradient step. μP guarantees that these two conditions are satisfied at any training step t. Roughly speaking, μP specifies that hidden weights should be initialized with $\Theta(n^{-1/2})$ random weights, and weight updates should be of order $\Theta(n^{-1})$. Input weights should be initialized $\Theta(1)$ and the weights update should be $\Theta(1)$ as well. While the output weights should be initialized $\Theta(n^{-1})$ and updated with $\Theta(n^{-1})$. These rules ensure both stability and feature learning in the infinite-width limit, in contrast to standard parameterization (exploding features if the learning rate is well tuned), and kernel parameterizations (e.g. Neural Tangent Kernel parameterization where $\Delta Y_l = \Theta(n^{-1/2})$, i.e. no feature learning in the limit).

A.2. The Gamma Function ($\gamma[.]$)

In the theory of scaling of neural networks, one usually tracks the asymptotic behaviour of key quantities as we scale some model ingredient. For instance, if we scale the width, we are interested in quantifying how certain quantities in the network behave as width n grows large and the asymptotic notation becomes natural in this case. This is a standard approach for (principled) model scaling and it has so far been used to derive scaling rules for initialization (Schoenholz et al., 2017b), activation function (Hayou et al., 2019), network parametrization (Yang et al., 2023), amongst other things.

With `Init[1]` and `Init[2]`, the weights are initialized with $\Theta(n^{-\beta})$ for some $\beta \geq 0$. Assuming that the learning rates also scale polynomially with n, it is straightforward that preactivations, gradients, and weight updates are all asymptotically polynomial in n. It is therefore natural to introduce the Gamma function, and we write $v = \Theta(\gamma[v])$ to capture this polynomial behaviour. Now, let us introduce some elementary operations with the Gamma function.

Multiplication. Given two real-valued variables v, v', we have $\gamma[v \times v'] = \gamma[v] + \gamma[v']$.

Addition. Given two real-valued variables v, v', we generally have $\gamma[v + v'] = \max(\gamma[v], \gamma[v'])$. The only case where this is violated is when $v' = -v$. This is generally a zero probability event if v and v' are random variables that are not perfectly correlated, which is the case in most situations where we make use of this formula (see the proofs below).

A.3. Proof of Proposition 1

Proposition 1. [Inefficiency of LoRA fine-tuning] *Assume that LoRA weights are initialized with `Init[1]` or `Init[2]` and trained with gradient descent with learning rate $\eta = \Theta(n^c)$ for some $c \in \mathbb{R}$. Then, it is impossible to have $\delta_t^i = \Theta(1)$ for all i for any $t > 0$, and therefore, fine-tuning with LoRA in this setup is inefficient.*

Proof. Assume that the model is initialized with Init[1]. Since the training dynamics are mainly simple linear algebra operation (matrix vector products, sum of vectors/scalars etc), it is easy to see that any vector/scaler in the training dynamics has a magnitude of order n^γ for some $\gamma \in \mathbb{R}$ (for more details, see the Tensor Programs framework, e.g. (Yang, 2019)). For any quantity v in the training dynamics, we write $v = \Theta(n^{\gamma[v]})$. When v is a vector, we use the same notation when all entries of v are $\Theta(n^{\gamma[v]})$. Efficiency is defined by having $\delta_i^t = \Theta(1)$ for $i \in \{1, 2\}$ and $t > 1$. Note that this implies $f_t(x) = \Theta(1)$ for all $t > 1$. Let $t > 1$ and assume that learning with LoRA is efficient. We will show that this leads to a contradiction. Efficiency requires that $\delta_t^i = \Theta(1)$ for all $t, i \in \{1, 2\}$. Using the elementary formulas from Appendix A.2, this implies that for all t

$$\begin{cases} \gamma[\eta] + 2\gamma[b_{t-1}] + 1 = 0 \\ \gamma[\eta] + 2\gamma[a_{t-1}^\top x] = 0 \\ \gamma[b_{t-1}] + \gamma[a_{t-1}^\top x] = 0. \end{cases}$$

Solving this equation yields $\gamma[\eta] = -1/2$, i.e. LoRA finetuning can be efficient only if the learning rate scales as $\eta = \Theta(n^{-1/2})$. Let us now show that this yields a contradiction. From the gradient updates and the elementary operations from Appendix A.2, we have the following recursive formulas

$$\begin{cases} \gamma[b_t] = \max(\gamma[b_{t-1}], -1/2 + \gamma[a_{t-1}^\top x]) \\ \gamma[a_t^\top x] = \max(\gamma[a_{t-1}^\top x], 1/2 + \gamma[b_{t-1}]) \end{cases}$$

Starting from $t = 1$, with Init[1] we have $\gamma[b_1] = \gamma[\eta(a_0^\top x)y] = -1/2$ and $\gamma[a_1^\top x] = \gamma[a_0^\top x] = 0$, we have $\gamma[b_2] = -1/2$ and $\gamma[a_2^\top x] = 0$. Trivially, this holds for any t. However, this implies that $\gamma[f_t] = \gamma[b_t] + \gamma[a_t^\top x] = -1/2$ which means that Δf_t cannot be $\Theta(1)$. With Init[2], we have $\gamma[b_1] = \gamma[b_0] = 0$ and $\gamma[a_1^\top] = \gamma[\eta b_0 y \|x\|^2] = -1/2 + 1 = 1/2$. From the recursive formula we get $\gamma[b_2] = 0$ and $\gamma[a_2^\top x] = 1/2$ which remains true for all t. In this case we have $\gamma[f_t] = 1/2$ which contradicts $\Delta f_t = \Theta(1)$.

In both cases, this contradicts our assumption, and therefore efficiency cannot be achieved in this setup.

□

A.4. Proof of Proposition 2

Proposition 2. [Efficient Fine-Tuning with LoRA] *In the case of Toy model Equation* (2)*, with* $\eta_a = \Theta(n^{-1})$ *and* $\eta_b = \Theta(1)$*, we have for all* $t > 1$, $\in \{1, 2, 3\}$, $\delta_t^i = \Theta(1)$.

Proof. The proof is similar in flavor to that of Proposition 1. In this case, the set of equations that should be satisfied so that $\delta_t^i = \Theta(1)$ are given by

$$\begin{cases} \gamma[\eta_a] + 2\gamma[b_{t-1}] + 1 = 0 \\ \gamma[\eta_b] + 2\gamma[a_{t-1}^\top x] = 0 \\ \gamma[\eta_a] + \gamma[\eta_b] + \gamma[b_{t-1}] + \gamma[a_{t-1}^\top x] + 1 = 0, \end{cases}$$

where we have used the elementary formulas from Appendix A.2. Simple calculations yield $\gamma[\eta_a] + \gamma[\eta_b] = -1$. Using the gradient update expression with the elementary addition from Appendix A.2, the recursive formulas controlling $\gamma[b_t]$ and $\gamma[a_t^\top x]$ are given by

$$\begin{cases} \gamma[b_t] = \max(\gamma[b_{t-1}], \gamma[\eta_b] + \gamma[a_{t-1}^\top x]) \\ \gamma[a_t^\top x] = \max(\gamma[a_{t-1}^\top x], \gamma[\eta_a] + \gamma[b_{t-1}] + 1). \end{cases}$$

Starting from $t = 1$, with Init[1], we have $\gamma[b_1] = \gamma[\eta_b(a_0^\top x)y] = \gamma[\eta_b]$ and $\gamma[a_1^\top x] = \gamma[a_0^\top x] = 0$. Therefore $\gamma[b_2] = \max(\gamma[\eta_b], \gamma[\eta_b] + 0) = \gamma[\eta_b]$, and $\gamma[a_2^\top x] = \max(0, \gamma[\eta_a] + \gamma[\eta_b] + 1) = \max(0, 0) = 0$. By induction, this holds for all $t \geq 1$. With Init[2], we have $\gamma[b_1] = \gamma[b_0] = 0$, and $\gamma[a_1^\top x] = \gamma[-\eta_a b_0^2 y \|x\|^2] = \gamma[\eta_a] + 1$. At step $t = 2$, we have $\gamma[b_2] = \max(0, \gamma[\eta_b] + \gamma[\eta_a] + 1) = 0$ and $\gamma[a_2^\top x] = \max(\gamma[\eta_a] + 1, \gamma[\eta_a] + 0 + 1) = \gamma[\eta_a] + 1$, and this holds for all t by induction. In both cases, to ensure that $\gamma[f_t] = \gamma[b_t] + \gamma[a_t^\top x] = 0$, we have to set $\gamma[\eta_b] = 0$ and $\gamma[\eta_a] = -1$ (straightforward from the equation $\gamma[\eta_b] + \gamma[\eta_a] = -1$). In conclusion, setting $\eta_a = \Theta(n^{-1})$ and $\eta_b = \Theta(1)$ ensures efficient fine-tuning with LoRA.

□

A.5. Proof of Theorem 1

In this section, we give a non-rigorous but intuitive proof of Theorem 1. The proof relies on the following assumption on the processed gradient g_A.

Assumption 1. *With the same setup of Section 4, at training step t, we have $g_A^t \underline{Z} = \Theta(n)$.*

To see why Assumption 1 is sound in practice, let us study the product $g_A^t \underline{Z}$ in the simple case of Adam with no momentum, a.k.a SignSGD which is given by

$$g_A = \text{sign}\left(\frac{\partial \mathcal{L}}{\partial A}\right),$$

where the sign function is applied element-wise. At training step t, we have

$$\frac{\partial \mathcal{L}_t}{\partial A} = \frac{\alpha}{r} B_{t-1}^\top d\bar{Z}^{t-1} \otimes \underline{Z},$$

Let $S^t = \frac{\alpha}{r} B_{t-1}^\top d\bar{Z}^{t-1}$. Therefore we have

$$g_A = \text{sign}(S^t \otimes \underline{Z}) = (\text{sign}(S_i^t \underline{Z}_j))_{1 \leq i,j \leq n}.$$

However, note that we also have

$$\text{sign}(S_i^t \underline{Z}_j) = \text{sign}(S_i^t)\text{sign}(\underline{Z}_j),$$

and as a result

$$g_A^t = \text{sign}(S^t) \otimes \text{sign}(\underline{Z}).$$

Hence, we obtain

$$g_A^t \underline{Z} = (\text{sign}(\underline{Z})^\top \underline{Z})\text{sign}(S^t) = \Theta(n),$$

where we used the fact that $\text{sign}(\underline{Z})^\top \underline{Z} = \Theta(n)$.

This intuition should in-principle hold for the general variant of Adam with momentum as long as the gradient processing function (a notion introduced in (Yang et al., 2013)) roughly preserves the $\text{sign}(\underline{Z})$ direction. This reasoning can be made rigorous for general gradient processing function using the Tensor Program framework and taking the infinite-width limit where the components of $g_A, \underline{Z}, d\bar{Z}$ all become iid. However this necessitates an intricate treatment of several quantities in the process, which we believe is an unnecessary complication and does not serve the main purpose of this paper.

Let us now give a proof for the main claim.

Theorem 1. *Assume that weight matrices A and B are trained with Adam with respective learning rates η_A and η_B and that Assumption 1 is satisifed with the Adam gradient processing function. Then, it is impossible to achieve efficiency with $\eta_A = \eta_B$. However, LoRA Finetuning is efficient with $\eta_A = \Theta(n^{-1})$ and $\eta_B = \Theta(1)$.*

Proof. With the same setup of Section 4, at step t, we have

$$\begin{cases} \delta_t^1 = B_{t-1}\Delta Z_A^t = -\eta_A B_{t-1} g_A^{t-1} \underline{Z} \\ \delta_t^2 = \Delta B_t Z_A^{t-1} = -\eta_B g_B^{t-1} A_{t-1} \underline{Z} \\ \delta_t^3 = \Delta B_t \Delta Z_A^t = \eta_A \eta_B g_B^{t-1} g_A^{t-1} \underline{Z} \end{cases}$$

The key observation here is that $g_A^{t-1}\underline{Z}$ has entries of order $\Theta(n)$ as predicted and justified in Assumption 1. Having $\delta_t^i = \Theta(1)$ for $i \in \{1, 2\}$ and $Z_B^t = \Theta(1)$ for $t > 1$ translate to

$$\begin{cases} \gamma[\eta_A] + \gamma[B_{t-1}] + 1 = 0 \\ \gamma[\eta_B] + \gamma[A_{t-1}\underline{Z}] = 0 \\ \gamma[B_{t-1}] + \gamma[A_{t-1}\underline{Z}] = 0, \end{cases}$$

which implies that $\gamma[\eta_A] + \gamma[\eta_B] = -1$.

With the gradient updates, we have

$$B_t = B_{t-1} - \eta_B g_B^{t-1}$$
$$A_t \underline{Z} = A_{t-1}\underline{Z} - \eta_A g_A^{t-1}\underline{Z}$$

which implies that

$$\gamma[B_t] = \max(\gamma[B_{t-1}], \gamma[\eta_B])$$
$$\gamma[A_t\underline{Z}] = \max(\gamma[A_{t-1}\underline{Z}], \gamma[\eta_A] + 1),$$

Now assume that the model is initialized with `Init[1]`. We have $\gamma[B_1] = \gamma[\eta_B]$ and therefore for all t, we have $\gamma[B_t] = \gamma[\eta_B]$. We also have $\gamma[A_1\underline{Z}] = \gamma[A_0\underline{Z}] = 0$ (because $A_1 = A_0$, and we use the Central Limit Theorem to conclude). Hence, if we choose the same learning rate for A and B, given by η, we obtain $\gamma[\eta] = -1/2$, and therefore $\gamma[Z_A^{t-1}] = \gamma[A_{t-1}\underline{Z}] = 1/2$ which violates the stability condition. A similar behaviour occurs with `Init[2]`. Hence, efficiency is not possible in this case. However, if we set $\gamma[\eta_B] = 0$ and $\gamma[\eta_A] = -1$, we get that $\gamma[B_t] = 0, \gamma[A_t\underline{Z}] = 0$, and $\delta_t^i = \Theta(1)$ for all $i \in \{1, 2, 3\}$ and $t \geq 1$. The same result holds with `Init[2]`.

□

B. Efficiency from a Loss Perspective.

Consider the same setup of Section 4. At step t, the loss changes as follows

$$\begin{aligned} \Delta\mathcal{L} &= \mathcal{L}((BA)_t) - \mathcal{L}((BA)_{t-1}) \\ &\approx \langle d\bar{Z}^{t-1} \otimes \underline{Z}, (BA)_t - (BA)_{t-1}\rangle_F \\ &= \langle d\bar{Z}^{t-1}, \Delta Z_B^t\rangle, \end{aligned}$$

where $\langle ., .\rangle_F$ is the Frobenius inner product in $\mathbb{R}^{n\times n}$, and $\langle ., .\rangle$ is the euclidean product in $\mathbb{R}^n$. Since the direction of the feature updates are significantly correlated with $d\bar{Z}^{t-1}$, it should be expected that having $\delta_t^i = \Theta(1)$ for all i results in more efficient loss reduction.

C. Additional Experiments

This section complements the empirical results reported in the main text. We provide the details of our experimental setup, and show the acc/loss heatmaps for several configurations.

C.1. Empirical Details

C.1.1. Toy Example

In Figure 2, we trained a simple MLP with LoRA layers to verify the results of the analysis in Section 3. Here we provide the empirical details for these experiments.

Model. We consider a simple MLP given by

$$f(x) = W_{out}\phi(BA\phi(W_{in}x)),$$

where $W_{in} \in \mathbb{R}^{n\times d}, W_{out} \in \mathbb{R}^{1\times n}, A \in \mathbb{R}^{r\times n}, B \in \mathbb{R}^{n\times r}$ are the weights, and ϕ is the ReLU activation function. Here, we used $d = 5$, $n = 100$, and $r = 4$.

Dataset. Synthetic dataset generated by $X \sim \mathcal{N}(0, I_d), Y = \sin(d^{-1}\sum_{i=1}^d X_i)$ with $d = 5$. The number of training examples is $N_{train} = 1000$, and the number of test examples is $N_{test} = 100$.

Training. We train the model with gradient descent for a range for values of (η_A, η_B). The weights are initialized as follows: $W_{in} \sim \mathcal{N}(0, 1.), W_{out} \sim \mathcal{N}(0, 1/n), A \sim \mathcal{N}(0, 1/n), B \sim \mathcal{N}(0, 1.)$. Only the weight matrices A, B are trained and W_{in}, W_{out} are fixed to their initial value.

C.1.2. GLUE Tasks with GPT2/Roberta

For our experiments with GPT2/Roberta-base models, finetuned on GLUE tasks, we use the following setup:

Tasks. MNLI, QQP, SST2, QNLI

Models. GPT2, Roberta-base

Training Alg. AdamW with $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon =$ 1e-8, linear schedule, no warmup.

Learning rate grid. $\eta_A \in \{$4e-3, 2e-3, 1e-3, 5e-4, 2e-4, 1e-4$\}$, $\eta_B \in \{$ 8e-4, 4e-4, 2e-4, 1e-4, 5e-5, 2e-5, 1e-5 $\}$.

Targert Modules for LoRA. For Roberta-base, we add LoRA layers to ‘query’ and ‘value’ weights. For GPT2, we add LoRA layers to ‘c_attn, c_proj, c_fc’.

Other Hyperparameters. Sequence length $T = 128$, train batch size $bs = 32$, number of train epochs $E = 3$ ($E = 10$ for SST2), number of random seeds $s = 3$.

GPUs. Nvidia V100, Nvidia A10.

C.1.3. Llama MNLI

For our experiments using the Llama-7b model, finetuned on MNLI, we use following setup

Training Alg. AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-6, constant schedule.

Learning rate grid. $\eta_A \in \{$1e-6, 5e-6, 1e-5, 2.5e-5, 5e-5, 1e-4$\}$, $\eta_B \in \{$1e-6, 5e-6, 1e-5, 2.5e-5, 5e-5, 1e-4$\}$, $\eta_B \geq \eta_A$

LoRA Hyperparameters. LoRA rank $r = 8$, $\alpha = 16$, and dropout 0.1. LoRA target modules ‘q_proj, k_proj, v_proj, o_proj, up_proj, down_proj, gate_proj’.

Other Hyperparameters. Sequence length $T = 128$, train batch size $bs = 32$, number of train epochs $E = 1$, number of random seeds $s = 2$ for $\eta_A = \eta_B$ and η_A, η_B near test optimal, $s = 1$ otherwise. Precision FP16.

GPUs. Nvidia V100.

C.1.4. Llama flan-v2

For our experiments using the Llama-7b model, finetuned on a size 100k random subset flan-v2, we use following setup

Training Alg. AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-6, constant schedule.

Learning rate grid. $\eta_A \in \{$1e-6, 5e-6, 1e-5, 2.5e-5, 5e-5, 1e-4$\}$, $\eta_B \in \{$1e-6, 5e-6, 1e-5, 2.5e-5, 5e-5, 1e-4$\}$, $\eta_B \geq \eta_A$

LoRA Hyperparameters. LoRA rank $r = 64$, $\alpha = 16$, and dropout 0.1. LoRA target modules ‘q_proj, k_proj, v_proj, o_proj, up_proj, down_proj, gate_proj’.

Other Hyperparameters. Sequence length $T_{\text{source}} = 1536$, $T_{\text{target}} = 512$, train batch size $bs = 16$, number of epochs $E = 1$, number of random seeds $s = 2$ for $\eta_A = \eta_B$ and η_A, η_B near test optimal, $s = 1$ otherwise. Precision BF16.

MMLU Evaluation. We evaluate average accuracy on MMLU using 5-shot prompting.

GPUs. Nvidia A10.

C.2. Results of Roberta-base Finetuning on all Tasks

Figure 3 showed finetuning test accuracy for Roberta-base. To complement these results, we show here the test/train accuracy for all tasks.

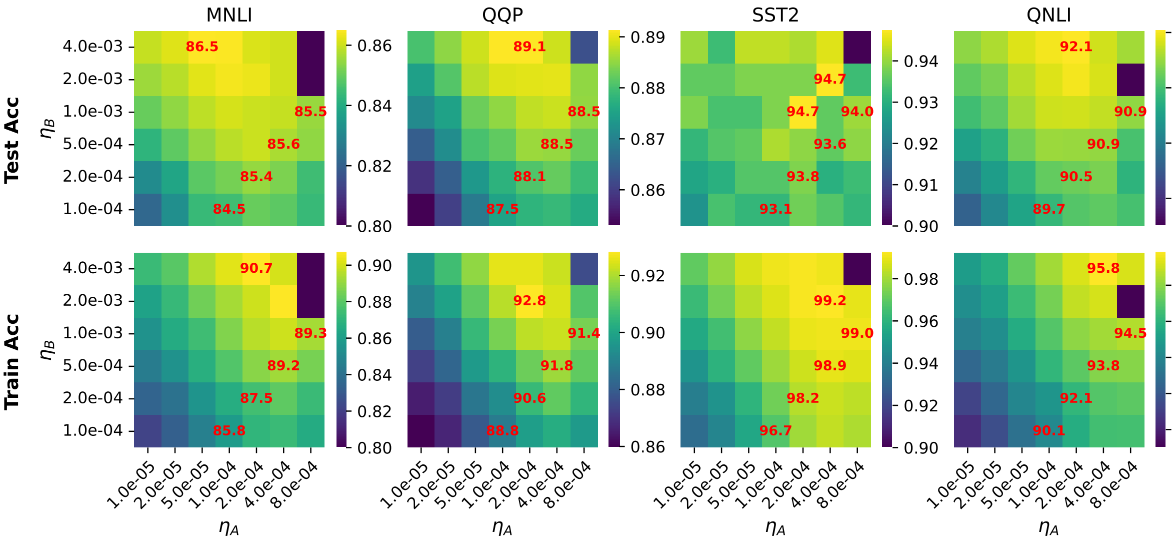


Figure 8. GLUE/Roberta-base: same as Figure 3 with test/train accuracy.

Interestingly, the optimal choice of learning rates for test accuracy differs from that of the train accuracy, although the difference is small. This can be due to mild overfitting occuring during finetuning (the optimal choice of learning rates (η_A, η_B) for train accuracy probably lead to a some overfitting).

C.3. Results of GPT2 Finetuning on all Tasks

Figure 4 showed finetuning results for GPT2 on MNLI and QQP. To complement these results, we show here the test/train accuracy for all tasks.

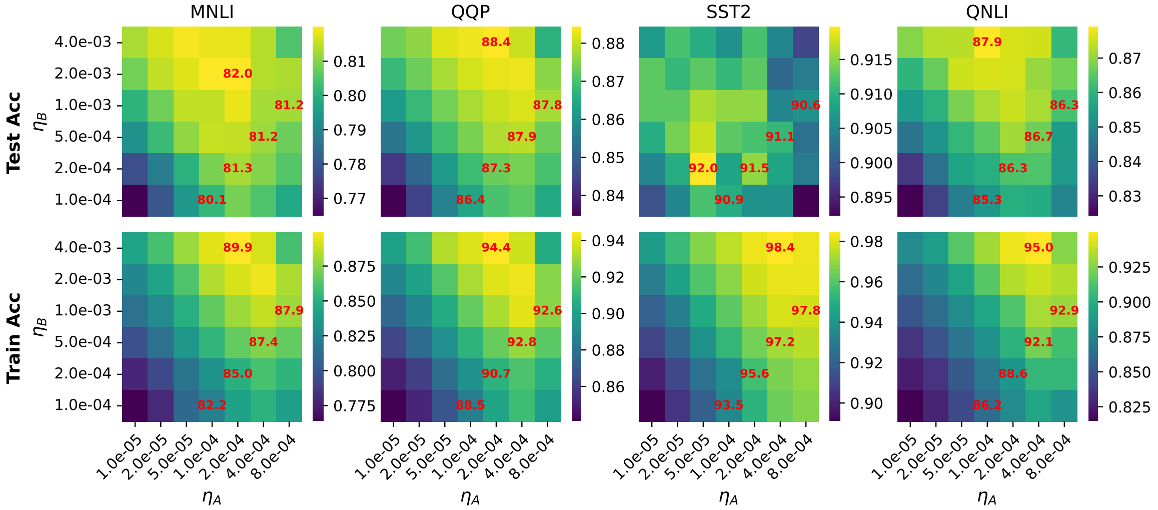


Figure 9. GLUE/GPT2: same setup as Figure 4 with additional tasks

C.4. GLUE Tasks with Full Precision

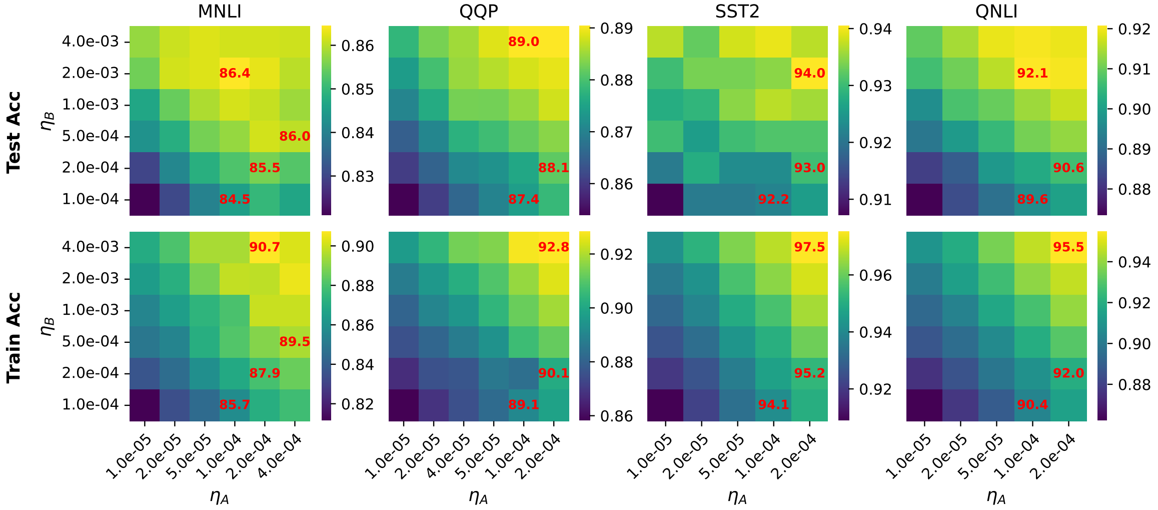


Figure 10. GLUE/Roberta-base: same as Figure 3 with full precision training instead of FP16.

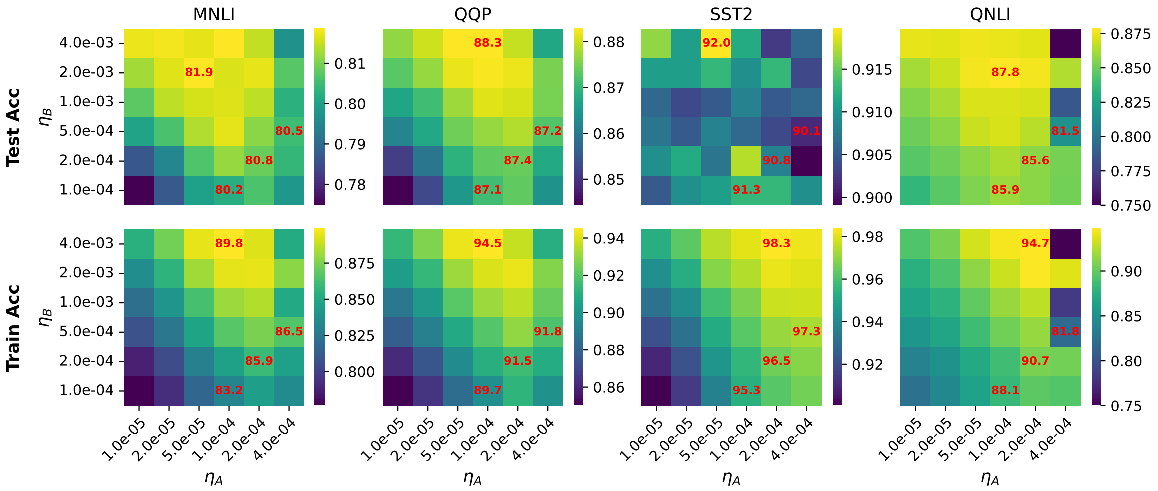


Figure 11. GLUE/GPT2: same setup as Figure 9 with full precision training

C.5. GLUE Tasks Test/Train Loss

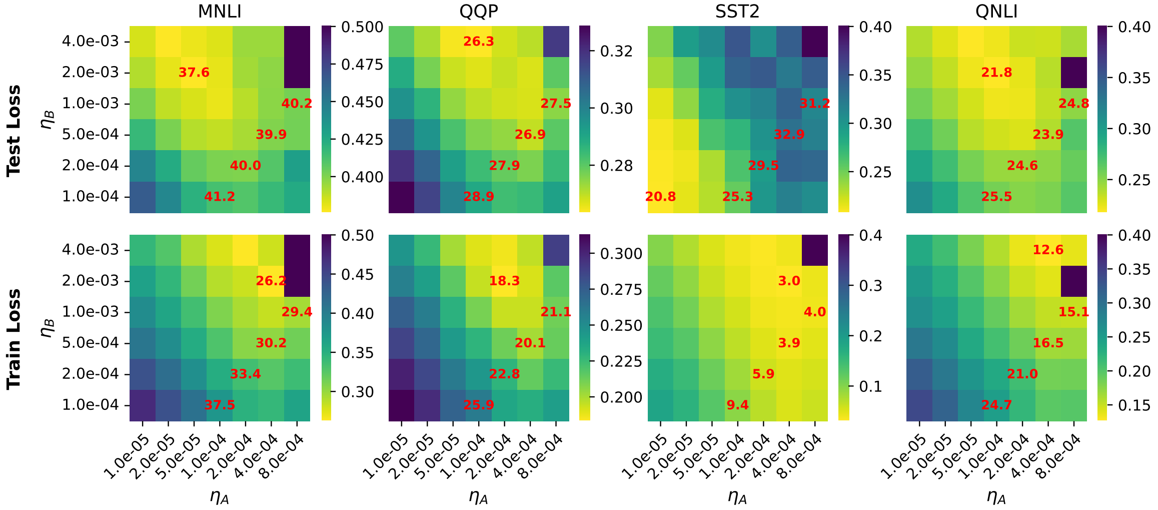


Figure 12. GLUE/Roberta-base: same setup as Figure 3 with 100×Test/Train loss instead of accuracy

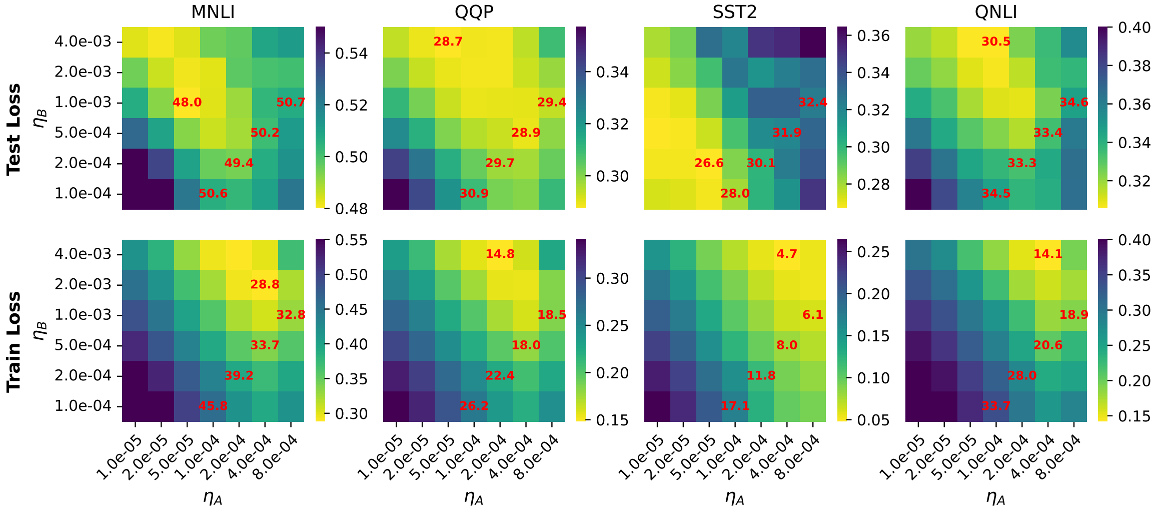


Figure 13. GLUE/GPT2: same setup as Figure 9 with 100×Test/Train loss instead of accuracy

C.6. GLUE Tasks with Different LoRA Ranks

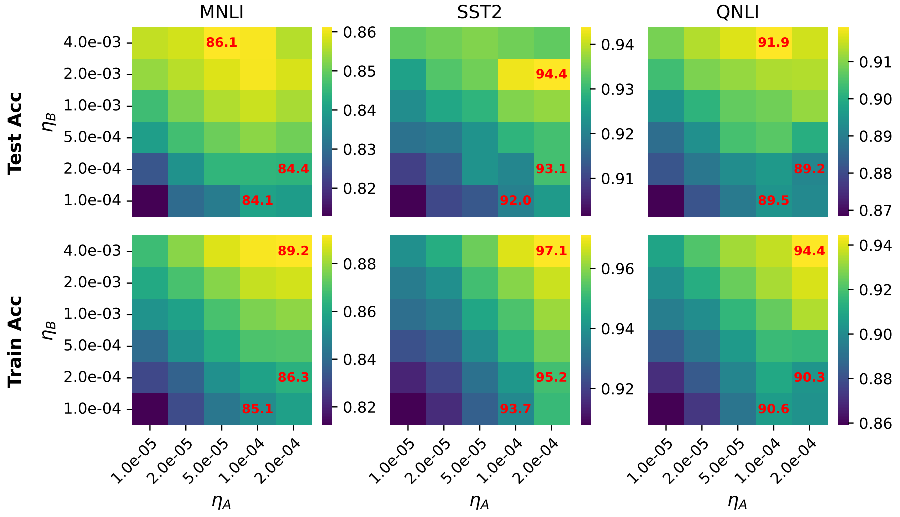


Figure 14. GLUE/Roberta-base: same setup as Figure 3 with $r = 4$

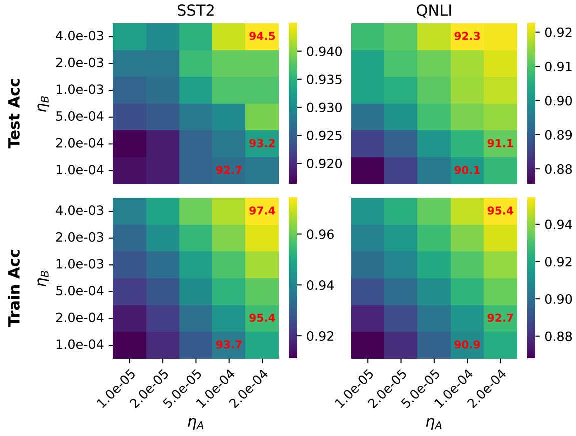


Figure 15. GLUE/Roberta-base: same setup as Figure 3 with $r = 16$

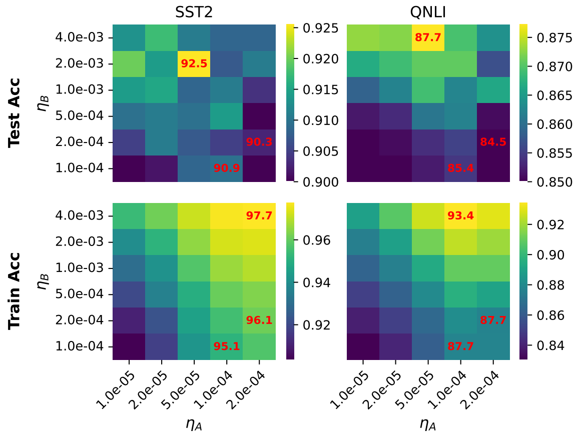


Figure 16. GLUE/GPT2: same setup as Figure 11 with $r = 4$

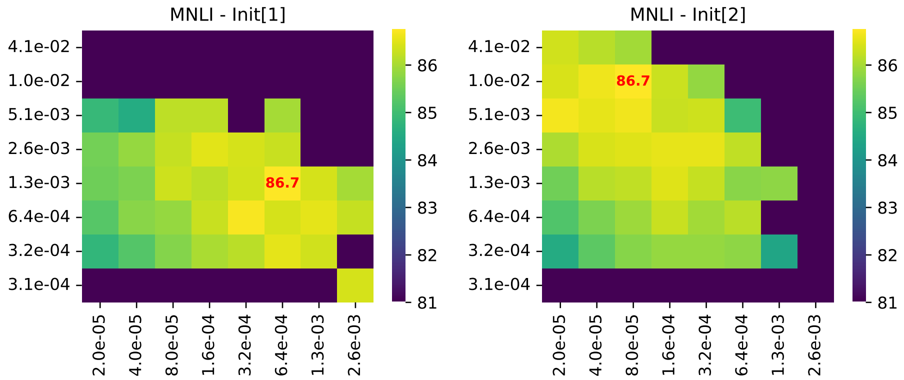
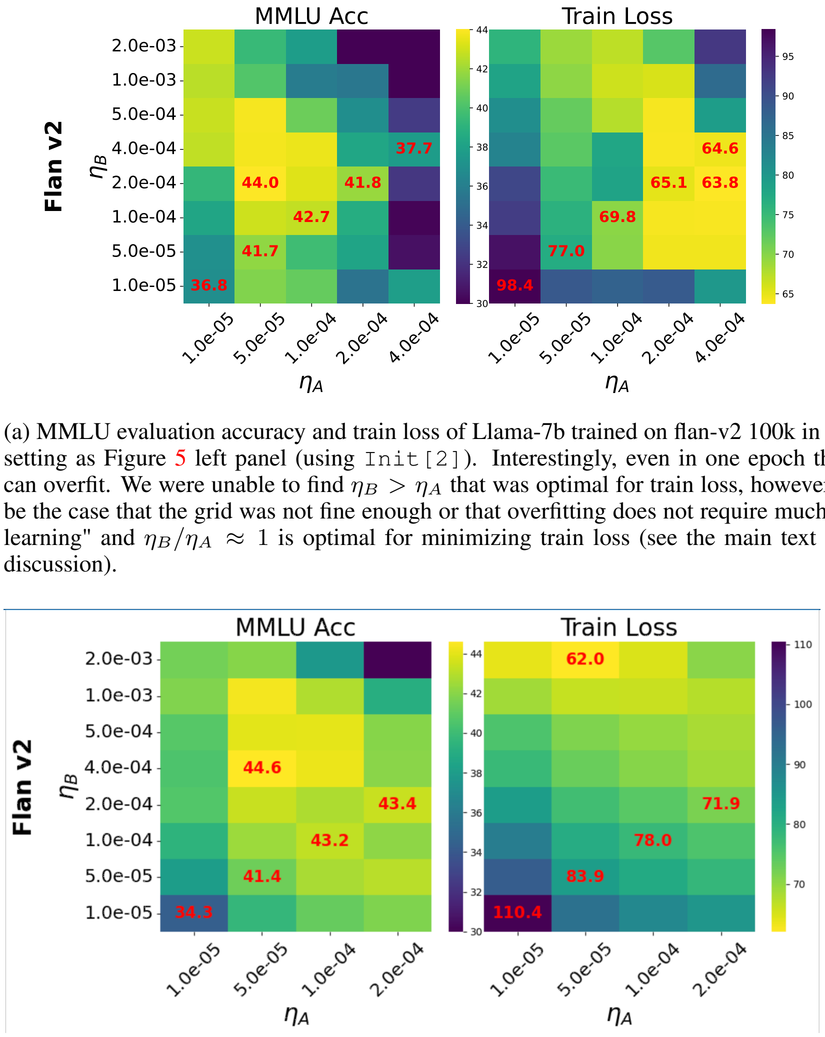


Figure 17. Roberta-base with `Init[1]` and `Init[2]`, finetuning on MNLI for 10 epochs (similar to Figure 3 but with more epochs).

C.7. Experiments with `Init[1]`

We also run some experiments using `Init[1]` as initialization scheme. We noticed that the optimal ratio λ is this case is generally smaller than the optimal ratio with `Init[2]`. Figure 17 shows the optimal learning rates (η_A, η_B) obtained with `Init[1]` and `Init[2]`. The optimal ratio $\lambda = \eta_B/\eta_A$ is generally smaller with `Init[1]`.

C.8. Llama Flan-v2 MMLU Acc/Train Loss



(a) MMLU evaluation accuracy and train loss of Llama-7b trained on flan-v2 100k in the same setting as Figure 5 left panel (using `Init[2]`). Interestingly, even in one epoch the model can overfit. We were unable to find $\eta_B > \eta_A$ that was optimal for train loss, however it could be the case that the grid was not fine enough or that overfitting does not require much "feature learning" and $\eta_B/\eta_A \approx 1$ is optimal for minimizing train loss (see the main text for more discussion).

(b) MMLU evaluation accuracy and train loss of Llama-7b trained on flan-v2 100k in the same setting as Figure 5 left panel except using `Init[1]`. Interestingly, the optimal MMLU accuracy is 0.6% higher than using `Init[2]` and the optimal ratio η_B/η_A is twice as large. The training loss is also near optimal only using a large ratio η_B/η_A.

Figure 18. Llama-7b on flan-v2 training with different initializations.

C.9. Llama MNLI Test/Train Loss

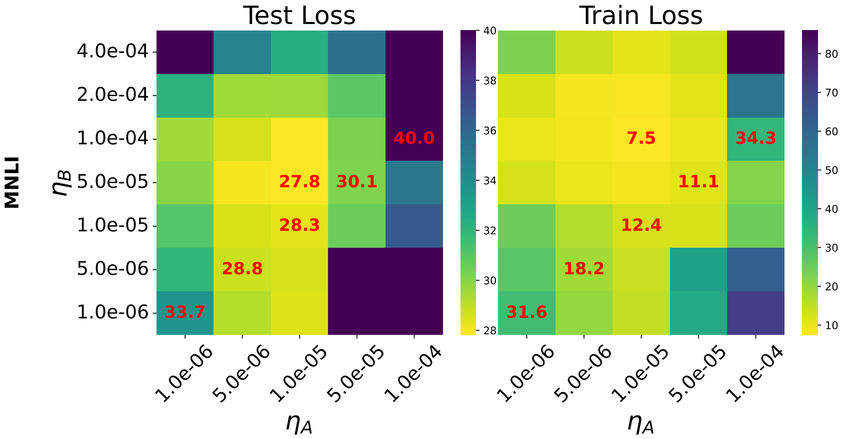


Figure 19. Train and test loss of Llama-7b finetuned on MNLI in the same setting as Figure 5 right panel.