LASSO: Enhancing GPT-2 with LoRA-Integrated AdamW, SMART, and SOAP Optimizers

Stanford CS224N Default

Tahmid Jamal
Department of Computer Science
Stanford University
tjamal8@stanford.edu

Julian Reed
Department of Computer Science
Stanford University
julianr@stanford.edu

Hugo Nathanael Yuwono
Department of Computer Science
Stanford University
hyuwono@stanford.edu

Abstract

As a result of transfer learning's rise in popularity, optimizers have become a fundamental aspect of increasing performances for large language models. However due to a constant scaling of these top-end models, optimizing every trillionth parameter has become computationally challenging and inefficient. We compare and contrast a range of optimization techniques to better understand how they can be combined for maximum performance increase without significant accuracy tradeoff. Specifically, we implement Low-Rank Adaptation (LoRA), a method of parameter-efficient finetuning (PEFT), and combine it with one of three optimizers: AdamW, SMART or SOAP. We evaluated the optimizers with LoRA and without it on two NLP tasks, accuracy for paraphrase detection and chrF score [1] for sonnet generation. We were able to achieve an accuracy of 0.882 on the test set for paraphrase detection and 41.772 chrF score on the test set for sonnet generation, both of which outperformed the baselines.

1 Key Information to include

- TA mentor: N/A
- External collaborators (if no, indicate "No"): No
- External mentor (if no, indicate "No"): No
- Sharing project (if no, indicate "No"): No

2 Introduction

Generative Pretrained Transformers (GPTs) have only been a part of our society within the last decade and it has become great priority in various sectors of the world and has become increasing crucial to create better, enhanced models. Since then, it has become a consensus that increasing the number of trainable parameters results in a model that achieves higher accuracy [2]. Therefore, it should not come as much of a surprise that a consistent development that we have witnessed over the last few years is the drastic increase in the number of learning parameters, which has soared from just over 100 million parameters in 2018, as seen in the base BERT model [3], to over 1.7 trillion parameters in 2023, as seen in OpenAI's ChatGPT 4's estimated number of parameters [4].

A predicament this trend has led to is implementing optimizers to fine-tune these LLMs. The earliest optimizer was Stochastic Gradient Descent [5] invented by Herbert Robbins and Sutton Monro in the

1950's, which parameters opposite to the gradients, and was the most revolutionary idea at the time. However, improving optimizers was a forgotten cause until the 50 years later in the 2010's when the deep learning revolution introduced many breakthrough optimizers. We see Adam and AdamW [6] make headlines in 2014, 2019 respectively, where Adam combined momentum with adaptive learning rates and AdamW added decoupling weight decay during optimization. However, these models struggle to keep the same efficiency with the scalability of these large models And so our motivation for this project is to enhance optimizers efficiency with other LLM techniques for NLP tasks.

In an attempt to solve the problem, we built a GPT-2 model and implemented Low-Rank Adaptation of Large Language Models (LoRA), Shampoo with Adam in the Preconditioner's Eigenbasis (SOAP), and Smoothness-Inducing Adversarial Regularization and Bregman Proximal Point Optimization (SMART) to solve paraphrase detection and sonnet generation tasks. Through LoRA, which introduces low-rank matrices that approximate weight updates to the query and value projection layers during fine-tuning to reduce computation, we attempt to solve the computational cost problem that has been plaguing the field of machine learning. Moreover, we also utilized LoRA in tandem with SOAP and SMART optimizers in order to achieve better performance compared in the aforementioned tasks compared to the baseline GPT-2 model.

3 Related Work

As previously stated, computationally efficient and accurate models are in high demand due to the success of large-scale language models. More parameters has resulted in better performance and thus newer models are consistently growing in size. However, this growth in size is accompanied by increased computational costs. In order to keep up with these increasing demands, several techniques had been developed in order to address these issues.

One avenue that has been research in detail, are Parameter Efficient Fine-Tuning (PEFT) methods in order to directly tackle the issue of the exponential rise of parameters for larger models. One technique, Intrinsic SAID [7], leverages the idea of intrinsic dimensionality to significantly decrease the number of trainable parameters. Intrinsic dimensionality refers to the concept of finding the minimum number of dimensions to solve a high dimensional task. Instead of minimizing the loss due to the original training, Intrinsic SAID reparameterizes the full model in a lower dimensional space and then computes the gradient by multiplying the current lower dimension gradient by a matrix $F : \mathbb{R}^r \to \mathbb{R}^d$ which is a Fastfood Transform. However an issue with this approach is that it doesn't scale to larger models and still requires training of up to $O(d)$ memory. To address this issues brought from Intrinsic SAID, Edward Hu et al. invented Low-Rank Adaption method (LoRA) [8] for these large language models. LoRA freezes the current GPT's parameters and then introduces 2 low-rank matrices that it uses to train on the model, and this has been proven to be significantly efficient with little to no accuracy decrease, motivating our selection of this method for our implementation of GPT2. A further enhanced extension is AdaLoRA [9], which dynamically changes the rank of the low rank matrices to control allocation. During updating the matrices, it introduces a diagonal matrix Λ which is iteratively reduced by truncating the smallest values. As a result, AdaLoRA significantly increases parameter and budgets allocation efficiency.

There has also been research on improving optimizers especially comparing among gradient descent algorithms. Gradient descent has been one of the most prominent and reliable optimizer in large language models with relative decent performance throughout, thus there are a lot of experiments improving and making variations to this optimizer. Stochastic Gradient Descent (SGD) [5] developed in 1950's by Robbins and Monro was revolutionary at the time because it performed gradient updates per training example thus making it faster. However it had issues of heavy fluctuation due to high variance updates and thus overcompensated for the weights each update. To address this issue, an extension of SGD, Adagrad[10] has adaptive learning parameters such that it performs greater updates to smaller parameters and less updates to bigger parameters, and smooths out the constant fluctuations. This allows for a more robust SGD and works effectively with even sparse data sets. Another interesting alley is updating your parameters based on estimates of future parameters. An interesting optimizer is Nesterov's Accelerated Gradient [11] where you estimate the future parameters and then update the current gradients with respect to future parameters.

It is clear that optimizers, as well was PEFT techniques, can bring large gains in model efficiency when implemented in isolation. Our paper aims to contribute an analysis of the benefits of combining these techniques (LoRA alongside SOAP, SMART or AdamW), specifically with the goal of maximally increasing model training speed while maintaining comparable levels of accuracy.

4 Approach

Our enhancements are built on top of a base GPT-2 model, which replicates that described in Radford et al. (2019) [12]. The novel technique we implement combines LoRA with each of the three optimizers (SOAP, AdamW, and SMART) individually, while we also implement each optimizer individually without LoRA to contextualize the benefits of using the two in tandem. Because LoRA's low rank matrices do not mandate any structural changes to the optimizers, we implemented each technique as if it were to be used in isolation. Each is fully described in more detail below.

4.1 Baselines

To understand the accuracy and efficiency benefits our approach provides, we will use the base Hugging-Face model which utilizes AdamW as our baseline for each of the optimizers. Then to measure the effect of LoRA on these optimizers, we will be using Hugging-Face's model to generate a baseline for each of the optimizer, then integrate LoRA to see the effect on performance. We will evaluate this base model on the Sonnet Generation and Paraphrase Detection tasks, recording both the model's accuracy and training time (iterations/second) as values to improve upon.

4.2 GPT-2 Model

The GPT-2 model is a decoder-only transformer model that utilizes the previous tokens to predict the next token. The GPT-2 model utilizes byte-pair encoding (BPE) tokenization, which greedily maximizing the number of tokens that get merged in a training set [13]. GPT-2 implements an embedding layer for each token and position embeddings for each position in the input. Its transformer layer is comprised of masked multi-head attention, skip connections, a multi-layer perceptron (MLP), and layernorm layers. Multi-head self-attention packs together the attention function on a set of queries where the attention function calculates the scaled dot product of a query. It is visualized by the following notation: $\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$ where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and $\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V$. In our model, we utilized masked multi-head attention, which adds an attention mask to the attention weights to get the post-mask weights, to prevent the tokens from attending to future positions. Finally, the model is trained to maximize the log-likelihood, which is illustrated by the following notation: $\log P(x_1, x_2, ..., x_n) = \log \Pi_{i=1}^n P(x_i|x_1, x_2, ..., x_{i-1}) = \Sigma_{i=1}^n \log P(x_i|x_1, x_2, ..., x_{i-1})$.

4.3 Low-Rank Adaptation of Large Language Models (LoRA)

Considering the fact that one of the most pervasive issues plaguing the field of machine learning is the computational cost required to train models [14], we decided to implement Low-Rank Adaptation of Large Language Models (LoRA), a PEFT method that introduces trainable low-rank matrices that approximate weight updates without modifying the model's original weights [8]. These matrices are then injected into the query (W_q) and value (W_v) projection layers of the self-attention mechanism. This reduces the computation required in the self-attention layer. Unlike conventional dimensionality reduction methods, LoRA does not reduce the computation cost by making alterations before the training of the model, but to the weight matrix during the fine-tuning process. It does so by freezing the pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$ and constraining its updates with a low-rank decomposition $W_0 + \triangle W = W_0 + BA$ where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and $r \ll \min(d, k)$. LoRA first initializes A by using a random Gaussian initialization and initializes B as the zero matrix, which means that before training, $\triangle W = AB$ is zero. In order to ensure that the coordinates of $\triangle Wx$ is $\Theta(1)$, we scale $\triangle Wx$ by $\frac{1}{r}$.

Moreover, unlike the conventional fine-tuning, where the weight decay in A and B is similar to decaying back to the pre-trained weights, LoRA also provides regularization advantages, which

allows it to outperform vanilla fine-tuning [15]. Other advantages of implementing LoRA include reducing VRAM consumption, checkpoint time, and training time, which reduces the number of GPUs needed and I/O bottlenecks. Our code for LoRA was adapted from hugging face's LoRA. [16]

4.4 wShampoo with Adam in the Preconditioner's Eigenbasis (SOAP)

Shampoo with Adam in the Preconditioner's Eigenbasis (SOAP) serves as an optimizer by making several changes to Shampoo - changing the power of the 2 preconditioners during weight updates, utilizing scalar correction to per layer learning rates, and making use of dataset averages for the preconditioners instead of using values across time steps [17]. Overall, SOAP is proven to have shown superior performance compared to AdamW (Adam with Weight Decay) and Shampoo, though it suffers from diminished efficiency gains of second-order methods at smaller batch sizes. Our code for SOAP is adapted from the Github repository linked in Vyas et al. [17]

The pseudocode for a step of soap can be seen below:

Algorithm 1: SOAPSTEP

Input: The 4 matrices per layer - $L \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{n\times n}$, $V, M \in \mathbb{R}^{m\times n}$, the hyperparameters η, betas $= (\beta_1, \beta_2)$, epsilon ϵ, and the preconditioning frequency f

Sample batch B_t
$G \in \mathbb{R}^{m\times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
$G' \leftarrow Q_L^T G Q_R$
$M \leftarrow \beta_1 M + (1-\beta_1)G$
$M' \leftarrow Q_L^T M Q_R$
Run AdamW on G
$V \leftarrow \beta_2 V + (1-\beta_2)(G' \odot G')$
$N' \leftarrow \frac{M'}{\sqrt{\hat{V}_t}+\epsilon}$
$N \leftarrow Q_L N' Q_R^T$
$W \leftarrow W - \eta N$
$L \leftarrow \beta_2 L + (1-\beta_2) G G^T$
$R \leftarrow \beta_2 R + (1-\beta_2) G^T G$
if $t\%f = 0$ **then**
  $Q_L = \text{Eigenvectors}(L, Q_L)$
  $Q_R = \text{Eigenvectors}(R, Q_R)$

4.5 Adam with Weight Decay (AdamW)

The Adam optimizer utilizes an exponentially moving average of the gradients and elementwise squared gradients [18]. If we are to denote the gradient as m_t and the squared gradient v_t where the hyperparameters $\beta_1, \beta_2 \in [0, 1)$ control the decay rates of the moving averages and α is the global learning rate, we get the the following learning rate $\alpha_t = \alpha \cdot \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ and the update of parameters $\theta_t \leftarrow \theta_{t-1} - \alpha_t \cdot \frac{m_t}{\sqrt{v_t}+\hat{\epsilon}}$ at update step t.

The Adam with Weight Decay (AdamW) optimizer adapts the Adam optimizer, but decouples weight decay and loss-based gradient updates in Adam [19]. The parameter update looks like this: $\theta_t \leftarrow \theta_{t-1} - \eta_t(\alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} + \lambda \cdot \theta_{t-1})$ where λ is the best overall weight decay value.

4.6 Smoothness-Inducing Adversarial Regularization and Bregman Proximal Point Optimization (SMART)

Smoothness-Inducing Adversarial Regularization and Bregman Proximal Point Optimization (SMART) imposes explicit regularization to control the model complexity during fine-tuning [20]. By using the Bregman Proximal Point Optimization, SMART attempts to optimize $\min_\theta \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta)$ where the loss function $\mathcal{L}(\theta)$ is de-

fined as $\mathcal{L}(\theta) = \frac{1}{n}\Sigma_{i=1}^{n}\ell(f(x_i;\theta),y_i)$ and ℓ is the chosen loss function. Moreover, $\mathcal{R}_s(\theta) = \frac{1}{n}\Sigma_{i=1}^{n}\max_{||\tilde{x}_i - x_i||\leq\epsilon}\ell(f(\tilde{x}_i;\theta),f(x_i;\theta))$.

Instead of using the vanilla Bregman Proximal Point Optimization, SMART accelerates the Bregman proximal point method by introducing an additional momentum to the update. It causes the update to look like the following: $\theta_{t+1} = \text{argmin}_\theta \mathcal{F}(\theta) + \mu\mathcal{D}_{\text{Breg}}(\theta,\tilde{\theta}_t)$ where $\tilde{\theta}_t = (1-\beta)\theta_t + \beta\tilde{\theta}_{t-1}$ and $\mathcal{D}_{\text{Breg}}(\theta,\theta_t) = \frac{1}{n}\Sigma_{i=1}^{n}\ell_s(f(x_i;\theta),f(x_i;\theta_t))$. Our code implementation of SMART is adapted from. Our implementation of SMART is an adaptation of the code in the Github repository linked in [21].

5 Experiments

5.1 Data

We use the two provided datasets for finetuning our GPT-2 model on each of our downstream tasks.

For Sonnet Generation, we used the Sonnet Dataset, which consists of 154 sonnets written by Shakespeare. This dataset contains 143 sonnets as training datapoints, and 12 sonnets as test datapoints.

For Paraphrase Detection, we used the Quora Question Pairs dataset, which contains pairs of questions with labels indicating whether particular instances are paraphrases of one another. This dataset contains 141,506 training examples, 20,215 dev examples, and 40431 test examples.

5.2 Evaluation method

To evaluate our models, we will use the task-specific accuracy metrics described in the default handout, as well as timing data to measure efficiency. Timing data is measured in iterations per second.

To determine accuracy on the Sonnet Dataset, we condition our model on the first three lines of the sonnet, at which point the model will generate the remainder. The chrF metric is used to quantify the differences between the generated text and the original sonnet [1]. As a classification task, accuracy is the metric for the Quora Question Pairs dataset.

5.3 Experimental details

We first ran the optimizers: AdamW, SMART, and SOAP without LoRA configurations in order to acquire our baselines for our two tasks. Our learning rate for the optimizers using the base model was $1e-5$ and our dropout probability is 0.1, Hugging Face GPT's default dropout.

Our base training times for sonnet task were AdamW : 0.39 its/sec, SMART : 0.15 its/sec, SOAP : 0.465 its/sec. Then we applied LoRA for our sonnet task and our configuration were $r = 512, \alpha = 31000, d = 0.1$. We kept the same learning rates and dropout rate in order to keep consistency and our LoRA-intergrated training times were, LoRA + AdamW : 0.315 its/sec, LoRA + SMART : 0.11 its/sec, LoRA + SOAP : 0.32 its/sec.

Then for our paraphrase task we had our base training times were AdamW : 15.4 it/sec, SMART : 4.77 it/sec, SOAP : 16 it/sec. Then we applied LoRA for our sonnet task and our configuration were $r = 512, \alpha = 3875, d = 0.1$. We also kept the same learning rates and dropout and our LoRA-intergrated training times were, LoRA + AdamW : 15.6 it/sec, LoRA + SMART : 4.56 it/sec, LoRA + SOAP : 16.05 it/sec.

Our final r, α, d values were all derived after multiple rounds of testing, with this configuration resulting in the highest accuracy along each task.

5.4 Results

Here are the results that we obtained, which comprises of the model's performance on paraphrase detection and sonnet generation.

5.4.1 Paraphrase Detection

The model's performance on the paraphrase detection task can be seen in the table below: Table 1

Table 1: Model Performance on the Paraphrase Detection Task

LoRA	Optimizer	Dev Set Accuracy	Time (Iterations / Second)
No	AdamW	0.897	15.4
No	SOAPV2	0.895	16
No	SMART	0.896	4.77
Yes	AdamW	0.839	15.6
Yes	SOAPV2	0.869	16.05
Yes	SMART	0.870	4.56

5.4.2 Sonnet Generation

The model's performance on the sonnet generation task can be seen in the table below: Table 2

Table 2: Model Performance on the Sonnet Generation Task

LoRA	Optimizer	Dev Set Score	Time (Iterations / Second)
No	AdamW	39.915	0.390
No	SOAPV2	40.625	0.465
No	SMART	40.539	0.150
Yes	AdamW	41.988	0.315
Yes	SOAPV2	42.066	0.320
Yes	SMART	42.060	0.110

6 Analysis

We believed that including Parameter-Efficient-Fine-Tuning methods to optimizers would enhance their performance without disrupting the layers the optimizers update the gradients for. Thus we wished to see if LoRA would affect the performance for AdamW, SOAP, and SMART optimizers. We observed that this did indeed occur for the sonnet generation task and comparing the times of optimizers with LoRA integrated, with just the optimizers themselves, we observed a significant increase in performance accuracy. However, for the task of paraphrase detection we saw an overall decrease with LoRA integrated architecture compared to simply using the optimizers themselves.

We saw very different outcomes within our two tasks of paraphrase detection and sonnet generation. Within sonnet generation we saw a significant increase in accuracy with the use of LoRA + optimizers and also saw a decrease in the number of average iterations per second. One potential reason to the lower average time is that LoRA needs to initialize the matrices first and the optimizers also have to initialize a entire new state specific to LoRA parameters, thus the computational costs is much greater in earlier epochs. However, once the matrices are aligned we saw a faster convergence with LoRA, and this is because of the fewer parameters that LoRA has to compute compared to the optimizers by themselves who have to update the gradients for every single parameter. Therefore with more epochs we believe LoRA could in fact be faster and boast a higher accuracy.

Within paraphrase detection we saw the opposite results where LoRA + optimizers had a decrease in accuracy compared to the optimizers alone. There could many potential reasons for this. One possibility is that our hyperparameters were not fine-tuned enough, and so we didn't find the most optimal LoRA parameters. Another possibility is that LoRA is not best suited for this specific task. Paraphrase detection has numerous complex semantic relationships, and so by lowering the dimension, LoRA might fail to capture these relationships in a smaller dimensional space. Our $\frac{\alpha}{r}$ ratio for our LoRA configurations was also relatively high, so perhaps there was over scaling in the rank matrices causing fluctuations in our gradient updates.

There was also another notable trend we observed was that during training, we saw that SMART took significantly more time to train than our other optimizers AdamW, and SOAP. This is most likely because for our implementation of SMART we wrapped SMART around an initial Adam optimizer loss and therefore we were computing two losses each epoch. Thus our training time was exponential greater for SMART however it still yielded favorable accuracy outperforming AdamW and even SOAP at times.

7 Conclusion

7.1 Summary of Findings

In this study, we successfully implemented LoRA and utilized it in tandem with AdamW, SMART, and SOAP optimizers on paraphrase detection and sonnet generation tasks. Our results showcase how LoRA can be beneficial for certain models working on certain tasks, but may also be detrimental when we aim to solve other tasks. For one, we observed that the implementation of LoRA results in improved performance in sonnet generation, which can be attributed to the regularization advantages that LoRA, provides through its low-rank decomposition, which, in turn, helps prevent overfitting. On the other hand, it did worse compared to the baseline in paraphrase detection tasks as the low-rank decomposition done by LoRA leads to the loss of semantic relationships present in the sentences, which is detrimental for this particular task as it may lead to an increased number of false positives.

Furthermore, our implementation of LoRA is yet to optimize the computational costs utilized, which is shown by how the running times per iteration of ours models that utilize LoRA is not superior compared to their counterparts that do not utilize LoRA. That can be attributed to the hidden rank value we chose when we configured LoRA in our models. Moreover, both SOAP and SMART (with and without LoRA) did outperform AdamW in terms of model accuracy, though that is not without compromising its runtime. Those 2 observations are the result of how each step of SOAP and SMART both make a call to AdamW's step function in addition to their respective additional calculations.

7.2 Limitations

As of now, our study of the implementation of LoRA is limited to one hidden rank value and 2 tasks - paraphrase detection and sonnet generation. As a result, we are yet to know the full extent of LoRA's impact towards computational efficiency and accuracy as well as which fields its implementation can be considered prudent.

7.3 Future Work

As potential future work, we believe that exploring the hidden ranks for the LoRA configuration, coupled with varying batch sizes across different hidden rank values, will allow us to achieve the goal of minimizing computational cost and maximizing performance through LoRA. Moreover, due to how it can be seen that LoRA is successful in improving the performance of our model, we are also interested in exploring different variants of LoRA, such as LoRA+ [22], LoRA-FA [23], and AdaLoRA [9], and see whether further improvements can be found. Finally, we think that it is beneficial to implement LoRA on a vast array of tasks and determine where LoRA can augment the model and where the use of LoRA is detrimental.

Team contributions (Required for multi-person team)

Julian owned development/implementation of SOAP. Tahmid owned development/implementation of LoRA. Hugo owned development/implementation of SMART. All team members collaborated equally to the implementation of the base GPT-2 model and the writing of the paper.

References

[1] Maja Popović. chrf: character n-gram f-score for automatic mt evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[2] Yuchen Xia, Jiho Kim, Yuhan Chen, Haojie Ye, Souvik Kundu, Cong Callie Hao, and Nishil Talati. Understanding the performance and estimating the cost of llm fine-tuning. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pages 210–223. IEEE, 2024.

[3] Ashish Khetan and Zohar Karnin. schubert: Optimizing elements of bert. *arXiv preprint arXiv:2005.06628*, 2020.

[4] Ibomoiye Domor Mienye, Theo G Swart, and George Obaido. Xtremellms: Towards extremely large language models. *Preprints*, 2024.

[5] Herbert Robbins and Sutton Monro. A stochastic approximation method. In *The Annals of Mathematical Statistics*, 1951.

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

[7] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[8] 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. In *arXiv preprint arXiv:2106.09685*, 2021.

[9] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Journal of Machine Learning Research*, 2011.

[11] Aleksandar Botev, Guy Lever, and David Barber. Nesterov's accelerated gradient and momentum as approximations to regularised update descent. *arXiv preprint*, 2016.

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

[13] Martin Berglund and Brink van der Merwe. Formalizing bpe tokenization. *arXiv preprint arXiv:2309.08715*, 2023.

[14] Jichao Li, Xiaosong Du, and Joaquim RRA Martins. Machine learning in aerodynamic shape optimization. *Progress in Aerospace Sciences*, 134:100849, 2022.

[15] Jiajun Hu, Jian Zhang, Lei Qi, Yinghuan Shi, and Yang Gao. Learn to preserve and diversify: Parameter-efficient group with orthogonal regularization for domain generalization. In *European Conference on Computer Vision*, pages 198–216. Springer, 2024.

[16] Hugging Face. Peft: Parameter-efficient fine-tuning of large language models - lora configuration. `https://github.com/huggingface/peft/blob/v0.14.0/src/peft/tuners/lora/config.py#L123`, 2023. Accessed: March 2025.

[17] Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and Stabilizing Shampoo Using Adam. In *arXiv preprint arXiv:2409.11321*, 2024.

[18] Kingma Diederik. Adam: A method for stochastic optimization. *(No Title)*, 2014.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[20] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. SMART: Robust and Efficient Fine-Tuning for Pre-Trained Natural Language Models Through Principled Regularized Optimization. In *arXiv preprint arXiv:1911.03437*, 2019.

[21] Archinet AI. Smart pytorch, 2025. Accessed: 2025-03-13.

[22] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

[23] Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.