Faster Convergence for Transformer Fine-tuning with Line Search Methods

1st Philip Kenneweg
AG Machine Learning
University Bielefeld
Bielefeld, Germany
0000-0002-7097-173X

2nd Leonardo Galli
dept. name of organization (of Aff.)
RWTH Aachen University
Aachen, Germany
0000-0002-8045-7101

3rd Tristan Kenneweg
AG Machine Learning
University Bielefeld
Bielefeld, Germany
0000-0001-8213-9396

4th Barbara Hammer
AG Machine Learning
University Bielefeld
Bielefeld, Germany
0000-0002-0935-5591

***Abstract*—Recent works have shown that line search methods greatly increase performance of traditional stochastic gradient descent methods on a variety of datasets and architectures [1], [2]. In this work we succeed in extending line search methods to the novel and highly popular Transformer architecture and dataset domains in natural language processing. More specifically, we combine the Armijo line search with the Adam optimizer and extend it by subdividing the networks architecture into sensible units and perform the line search separately on these local units. Our optimization method outperforms the traditional Adam optimizer and achieves significant performance improvements for small data sets or small training budgets, while performing equal or better for other tested cases. Our work is publicly available as a python package, which provides a hyperparameter-free pytorch optimizer that is compatible with arbitrary network architectures.**

***Index Terms*—Transformer, Stochastic Line Search, Optimization, BERT, NLP**

I. INTRODUCTION

In modern machine learning, there is a variety of optimization algorithms. Figuring out which one is the best and matching the learning rate or learning rate schedule to the specific problem can require a lot of expert knowledge and computing power. In particular, the current state of the art is to treat the learning rate as a hyperparameter and train the network until the value that yields the best performance is found, i.e., hyperparameter tuning. To simplify and significantly accelerate this process, a recent branch of the deep learning research [1], [3]–[5] has suggested to reintroduce line search methods for selecting the step size. These methods are widely adopted in the optimization community since their introduction in [6]. In particular, they automatically find an adaptive learning rate by calculating the loss at different points along the (gradient) direction and selecting an appropriate step size. Using line search can save time and resources by eliminating the need for manual optimization of the learning rate, thus eliminating costly hyperparameter tuning.

As traditional line search requires multiple forward passes per gradient update, a more efficient approach is needed. In [1], a Stochastic Line Search (SLS) has been combined with a smart re-initialization of the step size to alleviate the need for multiple forward passes for every step. This approach was shown to outperform a variety of other optimization methods, such as Stochastic Gradient Descent (SGD), SGD with Nesterov Momentum and many others on tasks such as matrix factorization as well as image classification for small networks and datasets.

It is however unclear if stochastic line search methods work well for larger networks and different architectures such as Transformers [7] as well as in challenging domains such as Natural Language Processing (NLP). In this work we adapt and expand upon this active topic of research in the NLP domain.

In addition, given the deep structure of recent networks, it is natural to ask if it would be beneficial to train each layer at a different speed, i.e. learning rate, instead of using a single value for the whole network. For this reason, we propose to modify SLS [1] and apply it in a layer-wise manner in order to specialize the step size for separate network components. This novel idea aims at localizing the line search in order to capture the loss variation due to the update of a single network component.

The Adam [8] optimizer has been shown to outperform SGD in the training of Transformers [**?**] by a large margin. To the best of our knowledge, this is the first time in the literature that SLS [2] in combination with Adam has been used to train Transformers or any other architecture. In particular, we will compare the following optimization schemas on the problem of fine-tuning Transformers:

- Adam optimizer with warm starting and cosine decay (ADAM).
- Armijo line search combined with stochastic gradient descent (SGDSLS).
- Armijo line search combined with the Adam optimizer (ADAMSLS).
- Per-layer-ADAMSLS, as a novel optimization method (PLASLS).

We test all optimizers fine-tuning BERT [9] on the Glue [10] dataset collection, which is widely used to evaluate common natural language processing skills. We find that SGDSLS does not work well in this scenario. In contrast, ADAMSLS and PLASLS perform better than ADAM, especially if they are trained on smaller data sets, as is often the case for retraining

We gratefully acknowledge the funding by the German Federal Ministry of Economic Affairs and Energy (01MK20007E).

language models [9]. To make our work easy to reproduce and easy to use, we implement all methods as pytorch optimizers. ADAMSLS can be used as an optimizer, that does not require to manually set the learning rate. For PLASLS the network parameters need to be manually split into components which are optimized separately.

In the next Section, we will review the related literature, in Section III we will describe our method. In Section IV we will describe our experimental approach and in Section V we will show and discuss our results. In Section VI we will go into details on some choices for PLASLS. Finally Section VII contains our conclusion.

II. RELATED WORK

The optimization of deep neural networks has been a central topic of research in the field of machine learning. A variety of techniques and optimizers have been proposed, including but not limited to SGD [11], Adagrad [12], RMSprop [13] and Adam [8]. One common challenge in optimizing neural networks is the selection of appropriate step sizes, which can have a significant impact on the convergence rate and final performance of a model [14].

Line search methods are a popular approach for selecting step sizes in optimization algorithms. These methods involve iteratively adjusting the step size based on the curvature of the loss function, in order to ensure that the objective is decreasing at each iteration. However, traditional line search methods can be computationally expensive, particularly for large neural networks with many layers.

In this work we particularly build upon [1]. In this paper, the authors show the theoretical proofs of convergence for the stochastic Armijo line search method in the case of strongly convex, convex and non convex functions. Furthermore, they provide solutions to some practical problems of line search methods. In particular, they no longer start the search from a fixed and possibly high initial step size, but instead from the last selected step size. To allow the step to also increase, they double it every m/b steps, where b is the minibatch size and m is a constant that does normally not need to be tuned, see Eq. 2. Additionally, they show empirical results on the image datasets MNIST [15], CIFAR10 [16] and CIFAR100 [16] using the ResNet [17] and DenseNet [18] architecture, as well as on a variety of convex problems.

Important advantages of the Armijo line search compared to other optimization methods are:

- no hyperparameter tuning of the learning rate
- faster convergence rates
- better generalization

However, their work does not show whether Armijo line search can outperform optimizers such as Adam on more complex tasks and architectures. To investigate this, we focus on natural language processing with Transformers, as this is a critical area of recent development and high complexity where other promising optimization methods have been unable to improve upon the baseline set by the Adam optimizer. Additionally, we investigate whether splitting the networks global step size into locally optimized step sizes is a viable option to further improve the performance of line search methods.

Recent work has shown that Transformers are very sensitive to the learning rate and learning rate schedule schedule they are trained with [19], [20]. To remedy this, various approaches like RADAM [20] or warm starting have been proposed. We show that our approach is able to train these highly sensitive architectures well.

Other recent work includes optimizing the learning rate simultaneously with the network weights [21]. The authors showed that this yields good results on classical image datasets, but does not seem to speed up convergence.

Further related work includes [22], which focuses on layer specific learning rates, [**?**] which studies the scaling limit of SGD in the high dimensional regime and [23] which studies why Adam is so effective at training the Transformer architecture.

Overall, the optimization of neural networks and transformer-based models remains an active area of research, and the development of efficient line search methods is an important contribution to this field.

III. METHODS

The classical stochastic Armijo line search [1] is designed to set a maximum step size for all network parameters w_k at iteration k (Eq. 1). In this section, together with the classical SGD direction, we will include the more complex Adam [8] direction. Moreover, we instead suggest to select a different step size for each different "component" of the network and evaluate the loss decrease w.r.t. it.

We define the following notation: The loss function is denoted by $f(w)$. $||\cdot||$ denotes the Euclidean norm and ∇f denotes the gradient of f. Given the iteration counter k, f_k and ∇f_k denote the mini-batch function and its mini-batch gradient.

A. Armijo Line Search

The Armijo line search criterion is defined in [1] as:

$$f_k(w_k + \eta_k d_k) \leq f_k(w_k) - c \cdot \eta_k ||\nabla f_k(w_k)||^2, \qquad (1)$$

where d_k is the direction (e.g., $d_k = -\nabla f_k(w_k)$ in case of SGD), $c > 0$ is a constant (commonly fixed to be 0.1 [**?**], [1], [2]). Condition 1 is practically obtained by employing a backtracking procedure, i.e., starting with an initial step-size η_k^0 and iteratively decreasing it by a constant factor $\delta \in (0, 1)$ until Eq. 1 is satisfied. As described in Section II, the following tweak was introduced in [1] to reduce the amount of backtracks, but avoid a monotonically decreasing step size

$$\eta_k = \eta_{k-1} \cdot 2^{b/m}. \qquad (2)$$

B. Including Adams Update Step in SLS (ADAMSLS)

In case of SGD, the direction d_k is the negative mini-batch stochastic gradient.

$$d_k = -\nabla f_k(w_k)$$

Adam's direction [8] can be defined as

$$\begin{aligned} g_{k+1} &= \nabla f_{ik}(w_k) \\ m_{k+1} &= \beta_1 \cdot m_k + (1-\beta_1) \cdot g_{k+1} \\ v_{k+1} &= \beta_2 \cdot v_k + (1-\beta_2) \cdot g_{k+1}^2 \\ \hat{m}_{k+1} &= m_k/(1-\beta_1^k) \\ \hat{v}_{k+1} &= v_k/(1-\beta_2^k) \\ d_k &= -\hat{m}_{k+1}/(\sqrt{\hat{v}_{k+1}}+\epsilon) \end{aligned} \tag{3}$$

Adam combines a momentum-based approach together with a step size correction built upon the gradients variance. In the training of Transformers, these modifications have been shown to be important enhancements over the simpler SGD [**?**]. The weight update rule is generally defined as

$$w_{k+1} = w_k + \eta_k d_k. \tag{4}$$

To apply the Armijo line search on the Adam optimizer we now use the direction d_k defined in Eq. 3, but with momentum $\beta_1 = 0$. Additionally, the gradient norm term $||\nabla f_k(w_k)||^2$ is replaced by the scaled gradient norm of Adam $\frac{||\nabla f_k(w_k)||^2}{\sqrt{\hat{v}_{k+1}}+\epsilon}$ resulting in Eq. 5.

$$f_k(w_k + \eta_k d_k) \leq f_k(w_k) - c \cdot \eta_k \frac{||\nabla f_k(w_k)||^2}{\sqrt{\hat{v}_{k+1}}+\epsilon}, \tag{5}$$

C. Layer Wise Line Search (PLASLS)

The key idea of layer wise line search is to split up the network parameters into L arbitrary sub-units, such as provided by network layers. In particular, we split w_k, d_k and the gradient $\nabla f_k(w_k)$ into their components and rewrite η_k as a vector:

$$\begin{aligned} w_k &= (w_k^{(1)}, \ldots, w_k^{(l)}, \ldots, w_k^{(L)}) \\ \nabla f_k(w_k) &= \left(\nabla f_k(w_k)^{(1)}, \ldots, \nabla f_k(w_k)^{(l)}, \ldots, \nabla f_k(w_k)^{(L)}\right) \\ d_k &= (d_k^{(1)}, \ldots, d_k^{(l)}, \ldots, d_k^{(L)}) \\ \eta_k &= (\eta_k^{(1)}, \ldots, \eta_k^{(l)}, \ldots, \eta_k^{(L)}). \end{aligned}$$

At this point, we adapt the Armijo line search to involve only elements of the sub-unit (l). However, the loss f is not partially separable [24], i.e., it can only be computed on the whole set of parameters w_k. Therefore, we define gradient components $\bar{d}_{k,l}$ and rewrite Eq. 1 as follows

$$\begin{aligned} f_k(w_k + \eta_k^{(l)} \bar{d}_{k,l}) &\leq f_k(w_k) - c \cdot \eta_k^{(l)} ||\nabla f_k(w_k)^{(l)}||^2, \\ \bar{d}_{k,l} &:= (0, \ldots, d_k^{(l)}, \ldots, 0) \end{aligned} \tag{6}$$

This not only reduces the computational costs of the line search, but also localizes the method to capture the variation of the loss only w.r.t. the l-th sub-unit. At each iteration k, we only apply a line search procedure on a single sub-unit to yield $\eta_k^{(l)}$, while all the other step sizes remains unchanged. In other words, if l is the sub-unit selected at iteration k the update can be rewritten as

$$\begin{aligned} w_{k+1} &= w_k + \eta_{k,l} \odot d_k, \\ \eta_{k,l} &:= (\eta_{k-1}^{(1)}, \ldots, \eta_k^{(l)}, \ldots, \eta_{k-1}^{(L)}), \end{aligned} \tag{7}$$

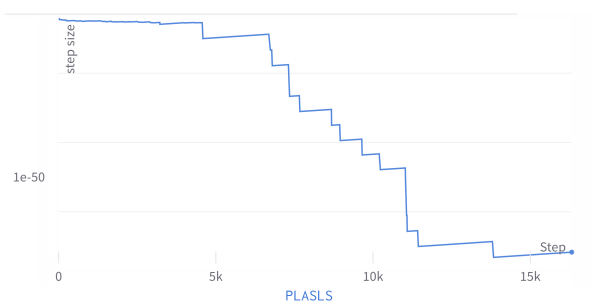


Fig. 1. Exemplary problematic step size of component 9 of the network during a single training run on the QNLI dataset. The step size starts out in the order of 10^{-4} but is lowered to values below 10^{-50}.

where $\odot$ is the Hadamard product, or component-wise product. Thus the resulting computational complexity is very similar to the original implementation, while allowing different step sizes for different network components.

Splitting: In the case of the Transformer [7] architecture or more specifically the BERT [9] architecture which is encoder only, we split the network into 10 different components. BERT has 12 encoder layers we divide the different layers into 8 different parts ($n_k^{(2)}$ affects layer 1, $n_k^{(3)}$ affects layers 2,3, $n_k^{(4)}$ affect layer 4, $n_k^{(5)}$ affect layers 5,6 ...). $n_k^{(1)}$ affects the embedding weights of the network. $n_k^{(10)}$ affects the single dense layer which is appended for a specific task. This is the optimal split found as shown by the experiments in Section VI.

First Step: Since the step size for each sub-unit is not changed during each iteration, a large initial step size η_0 could lead to divergence problems. To make our method completely automatic, we first perform a single step of SLS on the whole network to select a common first initial step size η_0. In particular, we choose a big initial step size $\eta_0 = 0.1$ and let the first line search find a good replacement for it. In fact, the step size yielded by the line search is a upper bound for the inverse of the Lipschitz constant of $\nabla f(w)$ (see Lemma 1 in [1]) and this provides us with a good approximation of the complexity of the problem at hand (at least in the starting point). From this value, we then start subdividing the step sizes per sub-unit.

Step Size Merging: If we look closely at the resulting step sizes yielded by our methods, we observe that they might converge to very low values (e.g., Figure 1). This also happens without the layer wise splitting, but less frequently. As a result, the affected part of the network is no longer able to learn. This can be desirable in some cases, but seems problematic overall.

In the specific case of layer-wise line search optimizer, a first solution for this problem would be to merge the network parts based on their current step size.

In Algorithm 1, we detect if a network components step size $n_k^{(l)}$ is below a certain threshold λ. If this is the case, this network component is merged with the network component with the second lowest step size. The new step size for this

Algorithm 1 algorithm for merging network components

step sizes $n_k^{(l)}$, threshold λ
find the smallest step size $n_k^{(s)} \in n_k^{(l)}$
if $n_k^{(s)} \leq \lambda$ **then**
  find the second smallest step size $n_k^{(s2)} \in n_k^{(l)}$
  merge network components s and $s2$
  $n_k^{(new)} = (n_k^{(s)} + n_k^{(s2)})/2$

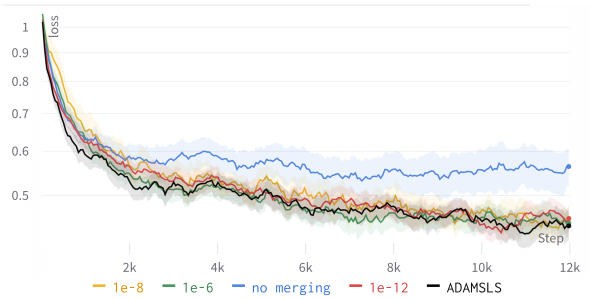


Fig. 2. Different merging thresholds on one epoch of the MNLI dataset. Standard error is indicated around each line.

combined network component is the average of both step sizes.

In Figure 2, the results of different merging thresholds λ are visualized, they seem to indicate that no merging performs worse than even the smaller merging thresholds of for example $\lambda = 10^{-12}$. Consequently, all experiments in Section IV for PLASLS are conducted using the merging threshold $\lambda = 10^{-12}$. With this threshold only very little automatic merging occurs. On the smaller datasets (all *small* datasets, MRPC) automatic merging does not occur, while on the larger datasets (SST2, QNLI, MNLI) on average about 3 of the 10 components are merged during a full 5 epoch training run. We strongly suspect that numerical cancellation errors are the cause of very low step sizes, we therefore plan to investigate this topic in more details in the future.

IV. EXPERIMENTAL APPROACH

In this section, we detail our experimental design to investigate the performance of the proposed optimizers. We utilize the Huggingface library [25] for implementation and the pre-trained Bert model 'bert-base-uncased' for all experiments.

A. Candidates

As a baseline comparison we evaluate the Adam optimizer with a peak learning rate of 2e-5 and a cosine decay with warm starting for 10% of the total training time, henceforth referenced to as $ADAM$. These values are taken from the original paper [9] and present good values for a variety of classification tasks, including the Glue [10] tasks upon which we are evaluating.

As another baseline, we use Armijo line search as described in [1] with stochastic gradient descent, we further refer to this approach as $SGDSLS$.

Next, we use the Armijo line search described in [1] and combine it with the Adam optimizer [8] as described in Section III to obtain the approach further referenced to as $ADAMSLS$.

The last option we are testing is the layer wise line search as described in Section III combined with the Adam optimizer further referenced to as $PLASLS$.

B. Implementation Details

The metaparameter choices for all experiments are as follows:

- All models are trained for 5 epochs.
- The pooling operation used in all experiments is [CLS].
- Batch size used for training is 32.
- The Adam optimizer with betas (0.9,0.999) and epsilon 10^{-8} is used.
- The maximum sequence length is set to 256 tokens.
- All models are trained 5 times and their mean metrics and standard error are reported.

C. Datasets

We consider a common scenario in natural language processing, where a large pre-trained language model is fine-tuned on a small dataset. The Glue dataset by Wang et al. [10] is a collection of various popular classification tasks in NLP, and it is widely used to evaluate common natural language processing capabilites. All datasets used are the version provided by tensorflow-datasets 4.0.1.

More specifically, we use the datasets Stanford Sentiment Treebank *SST2*, Microsoft Research Paraphrase Corpus *MRPC*, Stanford Question Answering Dataset *QNLI*, and Multi-Genre Natural Language Inference Corpus *MNLI*.

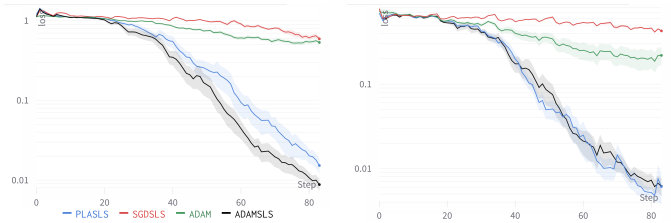
We evaluate all approaches on the *small* and full size datasets. The *small* datasets are the same as previously described, except that the size of the training dataset has been reduced to 500 randomly drawn samples. This scaling enables us to judge the capability of the optimizer in different dataset size regimes, as especially in real world scenarios small dataset sizes are a common occurrence.

V. EXPERIMENTAL RESULTS

In this section, we will describe the results of our experiments on the Glue dataset collection. We compare the 4 candidates as described in Section III

- ADAM
- SGDSLS
- ADAMSLS
- PLASLS

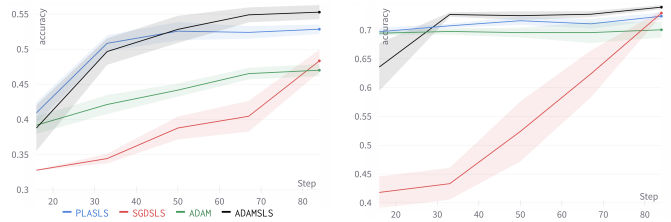
All accuracies displayed are the accuracies on the validation sets. The losses displayed are the losses calculated on the training sets, smoothed with exponential moving average. The colored areas around each line indicate the standard error of each experiment. We display the accuracies and losses during the training period in Figures 3,4,5,6. The Tables I and II are displaying the accuracies averaged over all datasets used, as is commonly done for the Glue dataset collection.



(a) MNLI small (b) MRPC small

(c) QNLI small (d) SST2 small

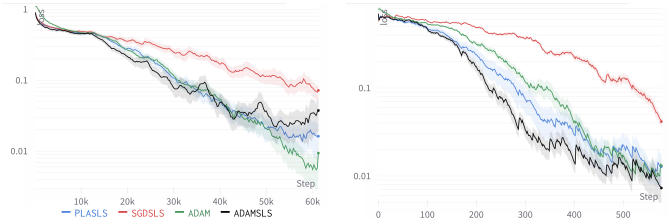
Fig. 3. The loss curves of the experiments on the small dataset with standard error indicated around each line. In all experiments SGDSLS performs the worst, followed by ADAM. PLASLS and ADAMSLS do not perform very different. In the SST2 and QNLI experiment PLASLS performs best, while in the MNLI experiment ADAMSLS performs best. In the MRPC experiments both perform about the same.



(a) MNLI small (b) MRPC small

(c) QNLI small (d) SST2 small

Fig. 4. The accuracy curves of the experiments on the small dataset with standard error indicated around each line, starting after the first epoch. In all experiments SGDSLS performs the worst, followed by ADAM. PLASLS and ADAMSLS do not perform very different. In the MNLI and MRPC experiment ADAMSLS performs best. In the SST2 and QNLI experiments ADAMSLS and PLASLS perform about the same.



(a) MNLI (b) MRPC

(c) QNLI (d) SST2

Fig. 5. The loss curves of the experiments on the full size dataset with standard error indicated around each line. Overall SGDSLS clearly performs worst. In the SST2 experiments PLASLS fails to converge to a very low loss. In the MRPC experiment we can see that ADAMSLS and PLASLS perform better initialy, but ADAM performs about the same in the end.

A. *Small Experiments*

In the small size experiments (see Figure 3 and Figure 4) we observe that the final loss of ADAMSLS and PLASLS is at least one order of magnitude lower compared to ADAM and SGDSLS. This is also reflected in the accuracy metric where PLASLS and ADAMSLS perform significantly better than ADAM or SGDSLS. Small losses on the training data and high accuracies on the validation data are highly correlated and no overfitting is observed.

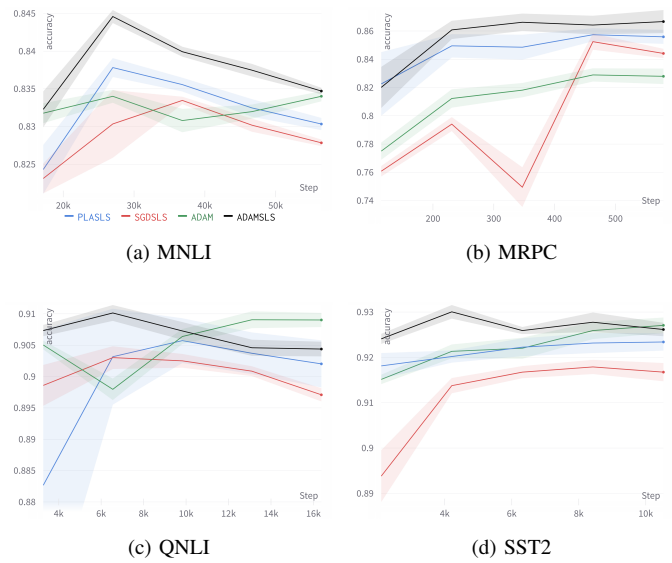
In Table I, we see the average performance over all datasets and runs at the end of training. ADAMSLS and PLASLS clearly outperform ADAM and SGDSLS by about 3%. Note that this accuracy is taken after 5 epochs, earlier in training even larger advantages for ADAMSLS and PLASLS can be observed.

TABLE I
AVERAGE CLASSIFICATION ACCURACIES, FOR THE *small* DATASETS. BEST PERFORMING METHOD IS MARKED IN **BOLD**.

method	ADAM	SGDSLS	ADAMSLS	PLASLS
accuracy	0.6875	0.6927	**0.7250**	0.7165

TABLE II
AVERAGE CLASSIFICATION ACCURACIES, FOR THE FULLSIZE DATASETS. BEST PERFORMING METHOD IS MARKED IN **BOLD**.

method	ADAM	SGDSLS	ADAMSLS	PLASLS
accuracy	0.8745	0.8714	**0.8830**	0.8779



(a) MNLI (b) MRPC

(c) QNLI (d) SST2

Fig. 6. The accuracy curves of the experiments on the full size dataset with standard error indicated around each line, starting after the first epoch. Accuracy was calculated on the validation data. Overall we can observe that SGDSLS performs the worst. ADAMSLS is overall the best performing optimizer.

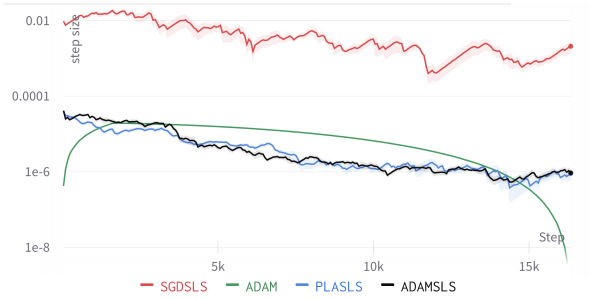


Fig. 7. Average step size of all layers during training runs on the QNLI dataset.

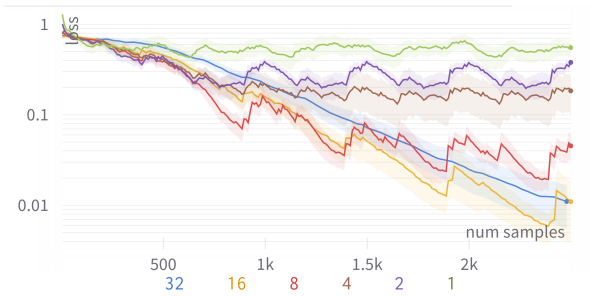


Fig. 8. Loss on the sst2small dataset for different batch size options for PLASLS. Standard error is visualized around each line. The X-Axis denotes the number of samples processed.

B. Full Size Experiments

In the full size experiments, we observe that the loss starts to decrease quicker with ADAMSLS and PLASLS. After about 10k steps ADAM outperforms the other options on the loss metric. On the accuracy metric we can see that ADAMSLS and PLASLS perform better in the beginning, but only slightly better at the end of training.

In Table II we see the final average performance over all datasets and runs. ADAMSLS and PLASLS very slightly outperform ADAM and SGDSLS by about 1%. It should be noted that these accuracies are measured after 5 epochs. Earlier in training, greater benefits can be observed for ADAMSLS and PLASLS.

C. Discussion

Overall we see that with smaller training sets or shorter training runs, ADAMSLS and PLASLS perform significantly better than ADAM or SGDSLS. This performance difference vanishes for longer training runs, here ADAM seems to perform better on the loss metric but still performs slightly worse in accuracy.

Interestingly, we can see in Figure 7 that the average step size of a well hyperparameter tuned ADAM and the automatically calculated ADAMSLS and PLASLS are similar during long periods of training. This demonstrates, the ability of ADAMSLS and PLASLS to result in very similar step size regimes to the tuned BERT steps size for the Adam optimizer even though they are not given an initial step size and thusly do not need any hyperparameter tuning.

VI. ABLATION STUDIES

In this section we will take a short look at which parts of our PLASLS approach have the most impact on performance, as well as different options for multiple elements of the described algorithm which we tested, but did not go into detail in the other sections.

A. Batch Size

During our experiments we noticed that the batch size had a large influence on the stability of the training. In this section we compare different batch sizes.

In Figure 8, we see that PLASLS performance improves for larger batch sizes during training. Furthermore, the loss curves get a lot smoother with higher batch sizes. During our experiments we choose a batch size of 32 as it was the highest supported batch size on our available hardware.

B. Split Options

Several possibilities to split a Transformer based network exist. In this section we look at a split by:

- layer
- query, key and value

We tried splitting the network in 1,4,7,10 components as well as by query,key and value (QKV). If we split the network in one component, ADAMSLS is equivalent to PLASLS. The loss curves for different layer configurations can be seen in Figure 9. More layers result in gradually better performance

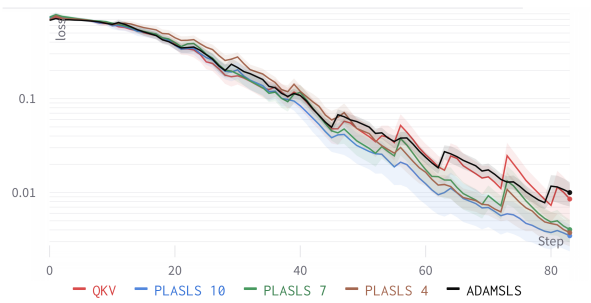


Fig. 9. Loss on the SST2small dataset for different split options for PLASLS and ADAMSLS as reference. Standard error is visualized around each line.

for the SST2small dataset. Splitting by QKV does not seem to improve performance.

VII. CONCLUSION

The type of optimizer, learning rate and learning rate schedule is a critical choice for every machine learning pipeline. We are the first in the literature to evaluate the combination of Adam and line search methods (ADAMSLS) and present a new variant of the line search optimizer PLASLS. Both optimizers we presented perform equal or better than state-of-the art baselines, especially on small data sets or short training runs in the domain of natural language processing for the ubiquitous Transformer architecture. We recommend using our ADAMSLS PyTorch implementation as best practice for the task of Transformer fine tuning.

For further work it would be interesting to combine the faster initial convergence rate of PLASLS or ADAMSLS with the long-term convergence of ADAM. Either by finding a more stochasticly stable replacement for the Armijo criterion, or by manually switching to the Adam optimizer after a certain amount of steps. By combining/improving the optimizer thusly, we could get an optimizer which would be even more widely applicable.

The source code is open-source and free (MIT licensed) software and available at
https://github.com/TheMody/Faster-Convergence-for-Transformer-Fine-tuning-with-Line-Search-Methods.

REFERENCES

[1] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien, "Painless stochastic gradient: Interpolation, line-search, and convergence rates," *NIPS'19: Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[2] S. Vaswani, F. Kunstner, I. H. Laradji, S. Y. Meng, M. Schmidt, and S. Lacoste-Julien, "Adaptive gradient methods converge faster with over-parameterization (and you can do a line-search)," *CoRR*, vol. abs/2006.06835, 2020.

[3] M. Mahsereci and P. Hennig, "Probabilistic line searches for stochastic optimization," *Advances in neural information processing systems*, vol. 28, 2015.

[4] R. Bollapragada, J. Nocedal, D. Mudigere, H.-J. Shi, and P. T. P. Tang, "A progressive batching l-bfgs method for machine learning," in *International Conference on Machine Learning*, pp. 620–629, PMLR, 2018.

[5] C. Paquette and K. Scheinberg, "A stochastic line search method with expected complexity analysis," *SIAM Journal on Optimization*, vol. 30, no. 1, pp. 349–376, 2020.

[6] L. Armijo, "Minimization of functions having lipschitz continuous first partial derivatives," *Pacific Journal of mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), pp. 4171–4186, Association for Computational Linguistics, 2019.

[10] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, (Brussels, Belgium), pp. 353–355, Association for Computational Linguistics, Nov. 2018.

[11] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.

[13] G. H. with Nitish Srivastava Kevin Swersky, "Lecture notes neural networks for machine learning," 2014.

[14] K. Nar and S. S. Sastry, "Step size matters in deep learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 3440–3448, Curran Associates Inc., 2018.

[15] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[19] P. Kenneweg, A. Schulz, S. Schröder, and B. Hammer, "Intelligent learning rate distribution to reduce catastrophic forgetting in transformers," in *Intelligent Data Engineering and Automated Learning – IDEAL 2022* (H. Yin, D. Camacho, and P. Tino, eds.), (Cham), pp. 252–261, Springer International Publishing, 2022.

[20] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *International Conference on Learning Representations*, 2020.

[21] K. Chandra, A. Xie, J. Ragan-Kelley, and E. Meijer, "Gradient descent: The ultimate optimizer," in *Advances in Neural Information Processing Systems* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.

[22] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, "Layer-specific adaptive learning rates for deep networks," 10 2015.

[23] F. Kunstner, J. Chen, J. W. Lavington, and M. Schmidt, "Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be," in *The Eleventh International Conference on Learning Representations*, 2023.

[24] L. Galli, A. Galligari, and M. Sciandrone, "A unified convergence framework for nonmonotone inexact decomposition methods," *Computational Optimization and Applications*, vol. 75, no. 1, pp. 113–144, 2020.

[25] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2019.