Stop Wasting My Time! Saving Days of ImageNet and BERT Training with Latest Weight Averaging

Anonymous Author(s)
Affiliation
Address
email

Abstract

Training vision or language models on large datasets can take days, if not weeks. We show that averaging the weights of the k latest checkpoints, each collected at the end of an epoch, can speed up the training progression in terms of loss and accuracy by dozens of epochs, corresponding to time savings up to ~68 and ~30 GPU hours when training a ResNet50 on ImageNet and RoBERTa-Base model on WikiText-103, respectively.

1 Introduction

The arsenal of deep learning methods (e.g., architectures, regularizers, pre-trainers, etc.) has been growing rapidly; for the last decade, thousands of them have been proposed yearly. Arguably, many are brittle and not as universally effective as initially claimed [24]. One way to filter *"what really works"* is by testing methods on large datasets. For example, for vision tasks, methods that demonstrated success on ImageNet have often proven to be successful in other tasks [1].

Large datasets, however, require access to expensive multi-GPU machines to enable data parallelism and reasonable training durations. Less well-funded researchers do not have access to supercomputers, and lengthy training runs make quick, iterative experimentation of research ideas difficult. Simple task-, model-, and optimizer-agnostic methods that can be easily added to existing training pipelines and speed up training time have the potential to make deep learning research more accessible.

In the 90s, Polyak & Juditsky [33] studied how to accelerate the convergence speed of stochastic gradient descent in the convex loss function regime. They proved that the running average of the model weights iterates $\bar{\boldsymbol{\theta}} = \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\theta}_i$ converges to the loss minimizer $\boldsymbol{\theta}^*$ asymptotically with the highest possible rate. When visualizing a convex loss function, the geometric intuition is simple: whenever the optimizer oscillates around a minimum, the average of the iterates will be closer to it.

However, in deep learning, loss functions are highly non-convex [8]. Weight averaging has been mainly used to improve the model's generalization performance at the end of or after training [14, 16].

Contribution We revisit weight averaging applied to neural networks from a convergence speed perspective. Inspired by Li et al. [22], we focus on the *middle* stage of training: after the dramatic changes of the local loss landscape during the very first training steps [10, 7] but *before* the optimizer converges. Because the weights still undergo substantial change in that middle phase, averaging *all* collected models, e.g., by maintaining a moving average [14, 16], can be sub-optimal. Therefore, we propose to average the k latest checkpoints (each collected at the end of an epoch) throughout training, which we refer to as *LAtest Weight Averaging* (LAWA).

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

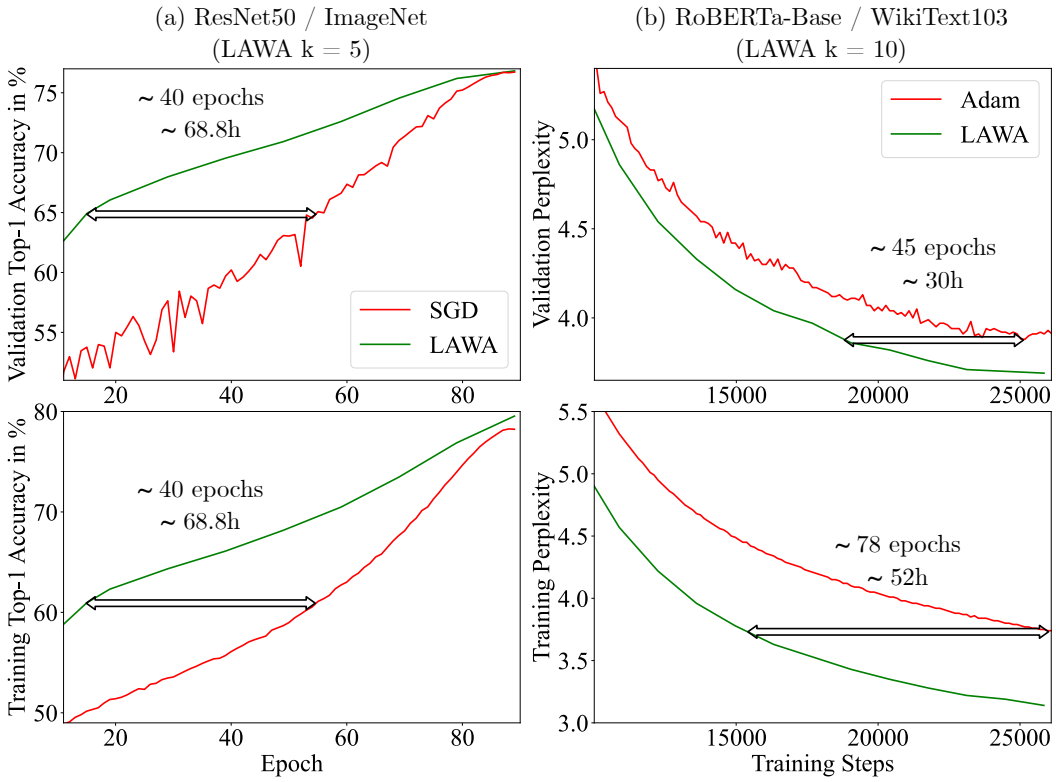


Figure 1: **LAWA speeds up convergence on ImageNet and (Ro)BERT(a) training**. We highlight the GPU hours saved by LAWA as the longest time until the baseline optimizer (SGD or Adam) matches LAWA 's performance. We plot the losses and top-5 accuracies in Appendix A.

2 LAtest Weight Averaging (LAWA)

The key idea is to collect model checkpoints once at the end of each epoch in a queue. LAWA 's solution at the end of epoch E is $\boldsymbol{\theta}_E^{\text{LAWA}} := \frac{1}{k}\sum_{i=E-k+1}^{E} \boldsymbol{\theta}_i$.

Requirements include few training loop modifications, as shown in Algorithm 1, and additional memory. In practice, we store the checkpoints in RAM or on disk and only transfer them to the GPU once we want to evaluate $\boldsymbol{\theta}^{\text{LAWA}}$.

The number of latest weights k is a hyper-parameter, and we achieve good results across both experiments with default value $k = 6$, as shown in Figure 2. However, we observe that averaging too many checkpoints ($k > 16$) results in worse performance.

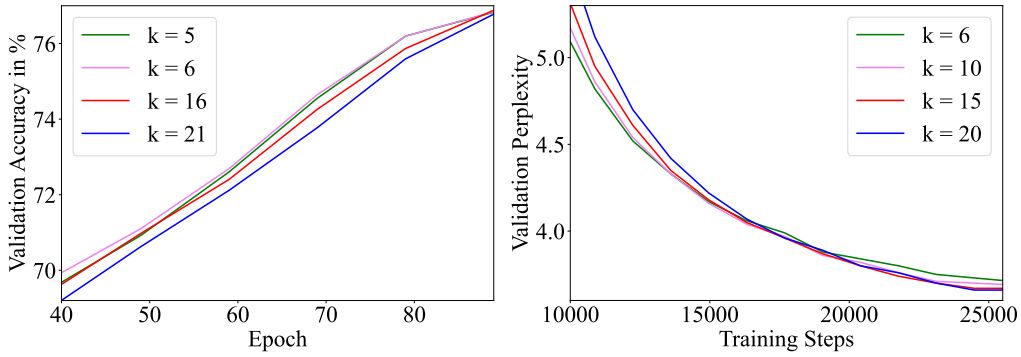
Algorithm 1 Pseudocode in PyTorch style

```
# k: number of latest checkpoints
ckpts = []
lawa_model = copy.deepcopy(model)
for epoch in range(num_epochs):
    for (x,y) in train_loader:
        train_step(x, y, model, optimizer)
    ckpts.append(get_params(model))
    if epoch + 1 >= k:
        update_lawa_model(lawa_model,
             ckpts)
        del ckpts[0]

def update_lawa_model(lawa_model, ckpts):
    for p, avg_p in zip(list(lawa_model.
         parameters()),torch.mean(ckpts)):
        p.copy_(avg_p)
```

The averaging coefficients can also follow a different pattern, e.g., in Appendix C, we experiment with an exponential moving average (assigning higher weights to the more recent checkpoints) and observe that this works worse than uniform coefficients. One can also learn the averaging coefficients [37, 22], but for simplicity and to avoid additional computational costs, we do not do so.

The checkpoint saving frequency might be thought of as a hyper-parameter; however, in this work, we always set it to one epoch. When there is so much data that we are in a sub-one-epoch training regime [18, 13], we may collect checkpoints every ν steps and need to tune ν. Another heuristic might be to collect a checkpoint whenever the validation loss has not improved [28].



(a) ResNet50 / ImageNet.

(b) RoBERTa-Base / WikiText103.

Figure 2: **LAWA is fairly insensitive w.r.t.** k, but high k can hurt its performance.

If the network includes batch norm layers, then their statistics for $\boldsymbol{\theta}_E^{\text{LAWA}}$ are unknown. Prior work [14] has suggested computing them by an inference pass through the training dataset. We do not observe a large effect of doing so compared to simply copying $\boldsymbol{\theta}_E$'s statistics, possibly because we only average the k latest weights instead of keeping one running average over many epochs [14].

3 Results

We run all experiments on a machine with 4x NVIDIA 3090s and report its wall-clock time.

3.1 Image Classification: ResNet50 on ImageNet

We consider the ImageNet 1000-classes classification task [4], which includes 1.28M training images and 50k validation images. To train a ResNet50 [12], we use the official PyTorch implementation [32] and train for 90 epochs using SGD with a momentum value of 0.9 and a cosine learning rate schedule. Our 4-GPU machine takes ~26min for one epoch. For $\boldsymbol{\theta}^{\text{LAWA}}$, we re-compute the batch norm layer statistics with a full inference pass through the training dataset before evaluating $\boldsymbol{\theta}^{\text{LAWA}}$.

In Figure 1(a), we observe that LAWA reaches a high accuracy dramatically faster, e.g., validation accuracy of around ~66% (the final accuracy is ~76%) is reached ~40 epochs (~68 hours) earlier than the baseline optimizer (SGD). However, we also note that its head start decreases towards the end of the training, and the highest reached accuracy is not reached much earlier. This observation raises the question of whether we can use LAWA to "jump forward" and continue the training from $\boldsymbol{\theta}^{\text{LAWA}}$ to reach the optimal accuracy faster, which we further discuss in Section 4.

3.2 Masked Language Modeling: RoBERTa-Base on WikiText-103

Next, we pre-train a (Ro)BERT(a)-Base [5, 25] model with masked language modeling objective on the WikiText-103 dataset [27] with 103M and 218k tokens for training and validation set, respectively. We follow the training recipe provided by `fairseq` [31]: We train with Adam [20] for 200 epochs, using a batch size of 2048, a polynomial learning rate decay with 10k warmup steps and a peak learning rate of 0.0005. Our 4-GPU machine needs ~10min for one training epoch.

In Figure 1(b), we report the training and validation perplexities as a function of the number of training steps (as typically done in NLP). We observe that LAWA consistently improves the validation perplexity, and it reaches Adam's final best validation accuracy ~45 epochs ahead, saving ~30 GPU hours. Interestingly, $\boldsymbol{\theta}^{\text{LAWA}}$'s final validation performance is noticeably better than $\boldsymbol{\theta}^{\text{ADAM}}$'s, confirming previous results on improved generalization obtained with weight averaging [14, 16].

4 Future Work

Continuing training from θ^{LAWA}. It is tempting to think that we may "jump forward" training by applying the LAWA procedure and then continue training from there if some target accuracy has not been reached yet. One issue is that we would need to adjust the learning rate each time we "jump". In practice, we may not know by how much (if at all) we accelerated the training progression. Hence, it remains unclear how to adjust a learning rate scheduler or the state variables of an adaptive optimizer.

k scheduler. In Figure 2, we observe that at different times, different k values perform better; e.g., during the end of the training, higher k performs better; motivating a scheduler for k.

Accelerating training from the very beginning. We focus on speeding up training during the middle stage of training: after the first training steps but long before the optimizer converges. The reason for that is that in the very early training phase, the gradient typically moves with large magnitudes until it converges to a smaller subspace of the loss function's Hessian, in which it then remains over long periods of training [10, 7] (middle stage). We empirically confirm that averaging during the early phase worsens the baseline's performance, as can be seen in Figure 4.

Combining LAWA with other acceleration techniques. As we will discuss in the next section, there are several other techniques available to accelerate neural network training. For example, the SAM optimizer [6] can accelerate training too [36], and Kaddour et al. [16] show that SAM combined with weight averaging can further boost the final test performance.

Relationships between LAWA and optimization hyper-parameters. For example, SGD becomes unstable for certain learning rates [38, 15]; can we similarly characterize when LAWA is effective?

Applying other operations to a set of checkpoints. For example, by learning a hyper-network [11] that takes in one or more checkpoints and predicts the model parameters at later training stages.

When does it not work? LAWA may not always cause speed-ups because Kaddour et al. [16] reported some negative results on using weight averaging to improve the model's final performance.

5 Related work

The idea of weight averaging is not novel; it has been studied widely in linear settings [33, 30, 21].

For the non-linear settings, Izmailov et al. [14] introduce *Stochastic Weight Averaging* (SWA), a weight averaging strategy starting from pre-trained models to move them to better-generalizing regions in the same loss basin. Kaddour et al. [16] extensively study SWA's effectiveness, including non-typical domains like graph-structured data, and suggest combining it with SAM [6] to boost its final performance further. Wortsman et al. [37] propose to average weights of multiple models with different hyper-parameter configurations. All three works (i) average weights toward the end or even after convergence, (ii) focus on the models' final test performances, and (iii) incorporate one moving-averaged model, while we show in Figure 2 that too large k can result in suboptimal results, especially at earlier training times.

This work is heavily inspired by Li et al. [22]'s *Trainable Weight Averaging* (TWA), who propose to learn averaging coefficients for training speed-ups. Concurrently, Guo et al. [9] observe that running the SWA procedure multiple times accelerates convergence. In some sense, LAWA generalizes their procedure by keeping an average of the k latest checkpoints instead of running SWA sequentially. Another related optimizer utilizing an auxiliary set of "fast weights" before updating the weights of interest is the *Lookahead* (LA) optimizer [40]. We compare LAWA and LA in Appendix B.

Another line of work has shown that training data re-weighting can speed up training. For example, some re-weighting methods focus on proxy models [2, 29], importance sampling [3, 19] or removing spurious correlations [17, 35].

References

[1] Beyer, L., Hénaff, O. J., Kolesnikov, A., Zhai, X., and Oord, A. v. d. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[2] Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., and Zaharia, M. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*, 2019.

[3] Csiba, D. and Richtárik, P. Importance sampling for minibatches. *The Journal of Machine Learning Research*, 19(1):962–982, 2018.

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[5] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=6Tm1mposlrM`.

[7] Frankle, J., Schwab, D. J., and Morcos, A. S. The early phase of neural network training. *arXiv preprint arXiv:2002.10365*, 2020.

[8] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Guo, H., Jin, J., and Liu, B. Stochastic weight averaging revisited. *arXiv preprint arXiv:2201.00519*, 2022.

[10] Gur-Ari, G., Roberts, D. A., and Dyer, E. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

[11] Ha, D., Dai, A., and Le, Q. V. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[12] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[13] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[14] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D. P., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. In Globerson, A. and Silva, R. (eds.), *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pp. 876–885. AUAI Press, 2018. URL `http://auai.org/uai2018/proceedings/papers/313.pdf`.

[15] Jastrzebski, S., Szymczak, M., Fort, S., Arpit, D., Tabor, J., Cho*, K., and Geras*, K. The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=r1g87C4KwB`.

[16] Kaddour, J., Liu, L., Silva, R., and Kusner, M. J. A fair comparison of two popular flat minima optimizers: Stochastic weight averaging vs. sharpness-aware minimization. *arXiv preprint arXiv:2202.00661*, 2022. doi: 10.48550/ARXIV.2202.00661. URL `https://arxiv.org/abs/2202.00661`.

[17] Kaddour, J., Lynch, A., Liu, Q., Kusner, M. J., and Silva, R. Causal machine learning: A survey and open problems. *arXiv preprint arXiv:2206.15475*, 2022. URL `https://arxiv.org/abs/2206.15475`.

[18] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[19] Katharopoulos, A. and Fleuret, F. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

[20] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Lakshminarayanan, C. and Szepesvari, C. Linear stochastic approximation: How far does constant step-size and iterate averaging go? In *International Conference on Artificial Intelligence and Statistics*, pp. 1347–1355. PMLR, 2018.

[22] Li, T., Huang, Z., Tao, Q., Wu, Y., and Huang, X. Trainable weight averaging for fast convergence and better generalization, 2022. URL `https://arxiv.org/abs/2205.13104`.

[23] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[24] Lipton, Z. C. and Steinhardt, J. Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*, 2018.

[25] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[26] Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Skq89Scxx`.

[27] Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016. URL `https://arxiv.org/abs/1609.07843`.

[28] Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[29] Mindermann, S., Brauner, J. M., Razzak, M. T., Sharma, M., Kirsch, A., Xu, W., Höltgen, B., Gomez, A. N., Morisot, A., Farquhar, S., et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pp. 15630–15649. PMLR, 2022.

[30] Neu, G. and Rosasco, L. Iterate averaging as regularization for stochastic gradient descent. In *Conference On Learning Theory*, pp. 3222–3242. PMLR, 2018.

[31] Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[32] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

[33] Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[34] Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[35] Tang, K., Huang, J., and Zhang, H. Long-tailed classification by keeping the good and removing the bad momentum causal effect. *Advances in Neural Information Processing Systems*, 33: 1513–1524, 2020.

[36] Team, T. M. M. composer. `https://github.com/mosaicml/composer/`, 2021.

[37] Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pp. 23965–23998. PMLR, 2022.

[38] Wu, L., Ma, C., et al. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. *Advances in Neural Information Processing Systems*, 31, 2018.

[39] Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[40] Zhang, M. R., Lucas, J., Ba, J., and Hinton, G. E. Lookahead optimizer: k steps forward, 1 step back. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 9593–9604, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/90fd4f88f588ae64038134f1eeaa023f-Abstract.html`.

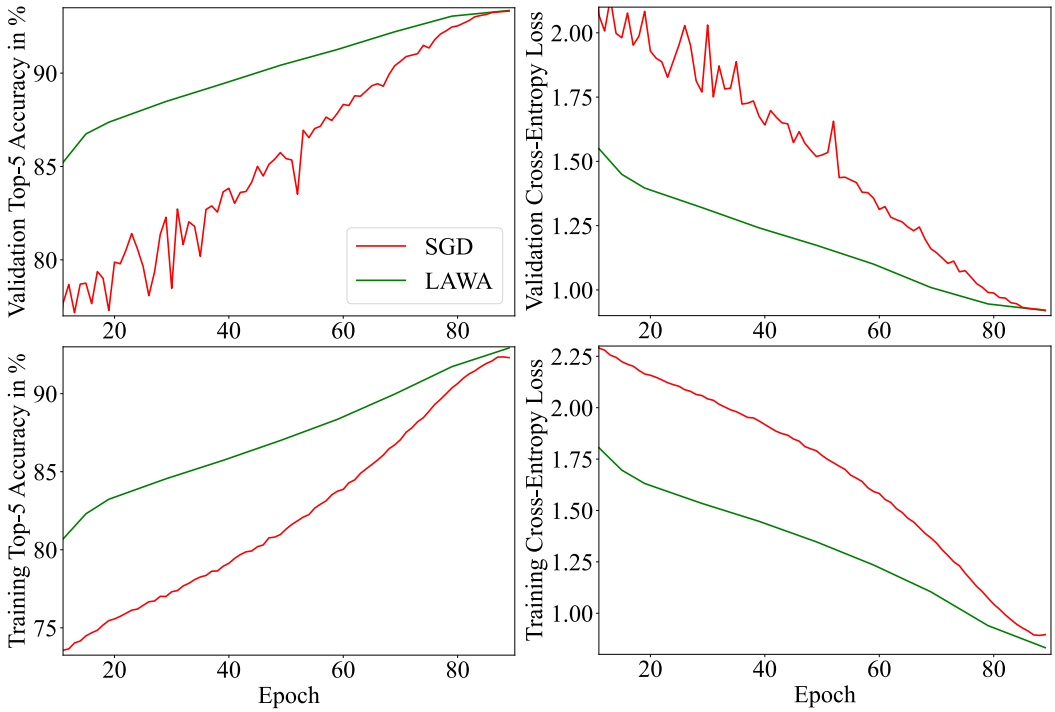


Figure 3: **Additional ResNet50 / ImageNet Metrics:** training and validation top-5 accuracies and losses.

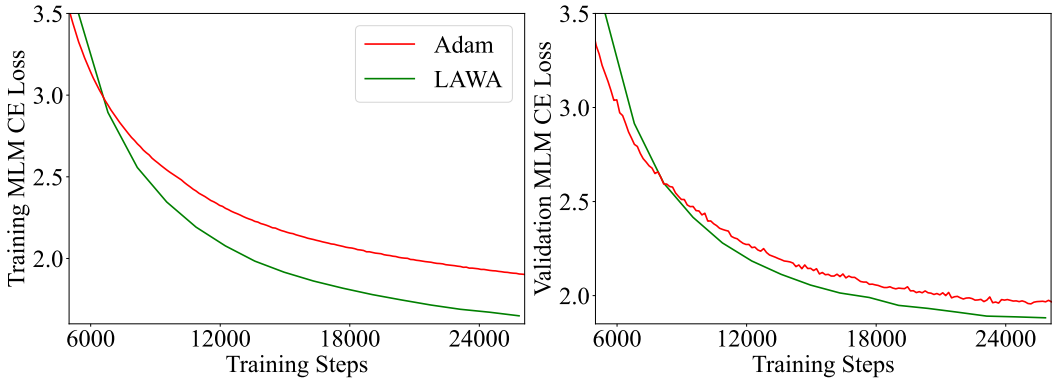


Figure 4: **Additional RoBERTa-Base / WikiText103 Metrics:** training and validation masked language modeling (MLM) cross-entropy (CE) [25] losses.

A Losses and Top-5 Accuracies

For completeness, we also plot the training and validation losses for both experiments and the top-5 accuracies for the ImageNet experiment.

Figure 3 shows similar speed up trends of LAWA over SGD as discussed in the main body (Figure 1).

Figure 4 shows the training and validation losses for RoBERTa-Base trained on WikiText103. Here, we also include losses during earlier stages of training and point out that during these more fluctuant phases, LAWA performs worse. We expect this behavior because previous works pointed out that the network undergoes dramatic changes in early phases [34, 10, 7].

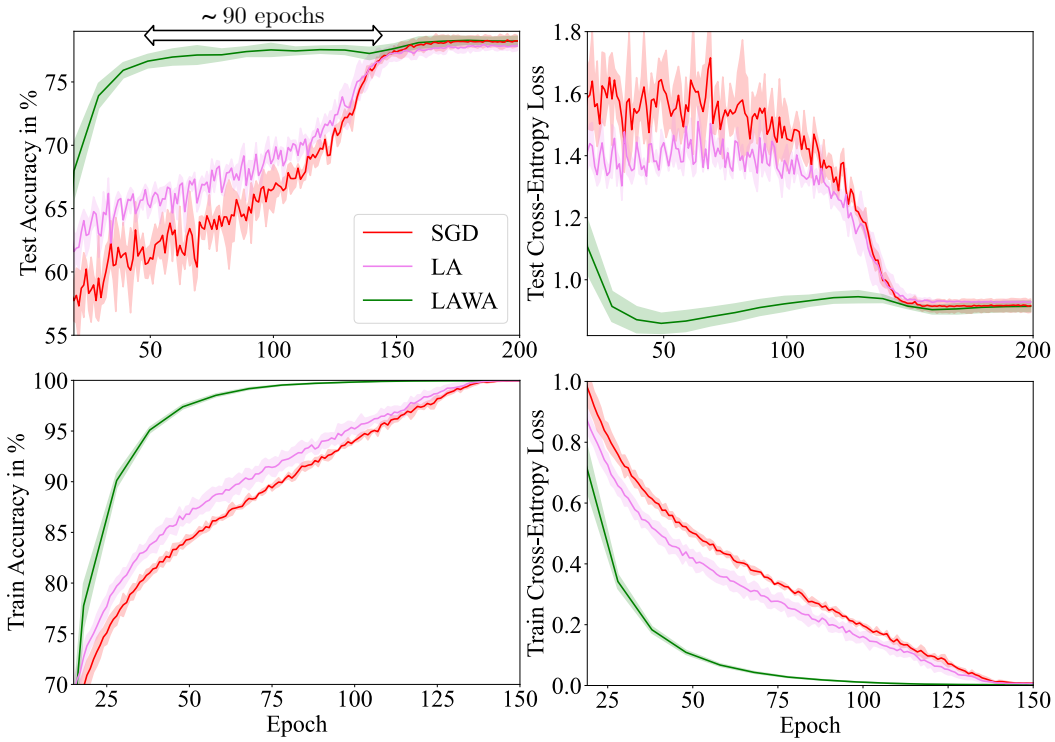


Figure 5: **LAWA ($k = 10$) outperforms Lookahead [40]**. We plot the mean and standard deviation of ResNet34/CIFAR-100 experiments across three random seeds.

B LAWA vs. Lookahead

We compare LAWA ($k = 10$) against Lookahead [40] on the moderately-sized CIFAR-100 dataset (50k training images) and train a ResNet34 [12]. We follow commonly used hyper-parameters (see e.g., [39, 23]), and train with SGD for 200 epochs, using a batch size of 256, a momentum value of 0.9 and a cosine learning rate scheduler [26, 6] with initial learning rate $\eta = 0.1$. For LA, we use $\alpha = 0.8$ and $k_{LA} = 5$, as suggested by the authors for this particular CIFAR100 dataset.

Figure 5 shows the training/test accuracy/loss as a function of the number of epochs. LAWA reaches high test accuracy around 90 epochs earlier than SGD/LA.

Initially, we started experimenting with this learning task before scaling up to larger datasets. Since we only observed slight but not dramatic improvements in LA over the baseline, we did not evaluate LA in the larger-scale ImageNet and BERT experiments. However, note that we apply LAWA to the SGD checkpoints; an interesting future direction can be to combine LAWA with LA, i.e., to average over checkpoints obtained with LA.

C Uniform vs. Exponentially Decayed Averaging Coefficients

We compare uniform (UNI, corresponding to $\boldsymbol{\theta}^{\text{LAWA}}$ by default) and exponentially-decaying (EXP) weight coefficients. We follow the same ResNet34 / CIFAR100 setup as in the previous section.

For EXP, we set $\alpha = 0.9$ and compute

$$\boldsymbol{\theta}_0^{\text{EXP}} = \boldsymbol{\theta}_0, \quad \forall E > 0 : \boldsymbol{\theta}_E^{\text{EXP}} := \alpha \boldsymbol{\theta}_E + (1 - \alpha) \boldsymbol{\theta}_{E-1}^{\text{EXP}}. \tag{1}$$

We set $k = 10$ for both strategies. Figure 6 shows that UNI slightly outperforms EXP; however, the difference is not large.

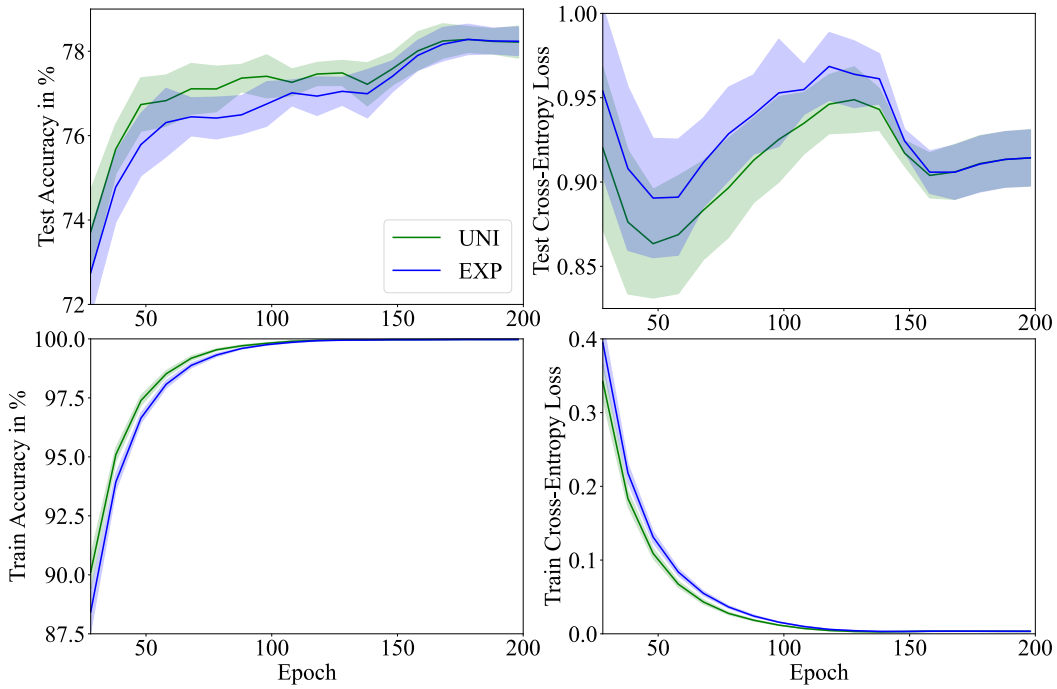


Figure 6: **Uniform coefficients (i.e., LAWA by default) slightly outperform exponentially-decaying ones for $k = 10$.** We plot the mean and standard deviation of ResNet34/CIFAR-100 experiments across three random seeds.