OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Heavy-tailed noise does not explain the gap between SGD and Adam on Transformers

Jacques Chen JACQUESC@STUDENTS.CS.UBC.CA
Frederik Kunstner KUNSTNER@CS.UBC.CA
Mark Schmidt SCHMIDTM@CS.UBC.CA
University of British Columbia

Abstract

The success of ADAM on deep learning architectures such as transformers has made it the default option in many application where stochastic gradient descent (SGD) does not work. Our theoretical understanding of this discrepancy is lagging and has prevented us from significantly improving either algorithm. Recent work advanced the hypothesis that ADAM, and other heuristics like gradient clipping, outperform SGD on language tasks because the distribution of the stochastic gradients is more heavy-tailed. This suggest that ADAM performs better because it is more robust to outliers, which is a promising avenue for designing better stochastic gradient estimators. However, it is unclear whether heavy-tailed noise is the cause of this discrepancy or another symptom. Through experiments that control for stochasticity by increasing the batch size, we present evidence that stochasticity and heavy-tailed noise is not the major factor in this gap.

1. Introduction

The success of ADAM (Kingma and Ba, 2015) and other heuristics on deep learning problems such as Transformers (Vaswani et al., 2017) has made them the default in those settings. They outperform stochastic gradient descent (SGD) by such a margin that SGD is considered incapable of training those architectures (Liu et al., 2020) and is omitted from performance comparisons (e.g. Anil et al., 2019). However, we only have a limited theoretical understanding of why those heuristics work.

Our usual problem classes, such as non-convex and smooth, or non-smooth and convex, do not capture the specific difficulties of training those architectures, and state-of-the-art, worst-case analyses do not capture improvements of ADAM over SGD. We have not yet been able to show, for example, that the moving averages lead to improvements (Défossez et al., 2020; Alacaoglu et al., 2020). There is a sentiment in the optimization community that the success of those heuristics is not due to their performance, but to social dynamics or a co-evolution of deep learning optimization heuristics and architectures (Orabona, 2020). However, it remains that ADAM-type algorithms outperform SGD on classes of problem that are relevant to the machine learning community, particularly in natural language processing. This suggest that those algorithms are adapted to the problem they are solving, and that there is some structure that our current theory—and theory-derived algorithms—is not using to the fullest extent. Understanding what this structure is may be the key to develop better practical algorithms in this setting.

A notable difference between some problems where SGD performs well and some where it lags behind was identified by Zhang et al. (2020). They show that for common architectures encountered in learning from images or text, the distribution of the error in stochastic gradients has heavier tails

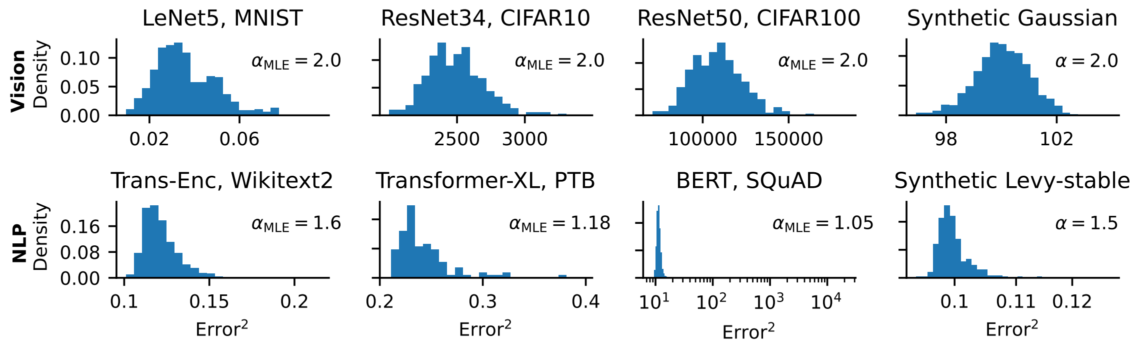

Figure 1: Stochastic gradient error $\|\tilde{g}(x) - \nabla f(x)\|^2$ across minibatches for image and text problems. Stochastic gradients on text data have heavier tail than those on image data, matching earlier observations. Synthetic Gaussian, Lévy alpha-stable distributions and estimates α_{MLE} of a Lévy-stable distribution fit to the errors for comparison.

on text problems, where we observe a difference in performance between SGD and heuristics like ADAM and clipping (Pascanu et al., 2013), as illustrated in Figure 1. They suggest that outliers in the distribution of the stochastic gradients could be the reason for the worse performance of SGD.

The thought experiment they provide for the extreme but simplified version of the hypothesis is a helpful illustration; suppose the distribution of the stochastic gradient $\tilde{g}$ for a function f is so heavy-tailed such as to have infinite variance, $\mathbb{E}[\|\tilde{g}(x) - \nabla f(x)\|^2] = \infty$. SGD will incur infinite expected error at each step. On the other hand, gradient clipping and ADAM use estimates of the gradient that, although biased, have finite error and can make progress. Practical problems will rarely follow such a worse-case distribution, especially if the stochasticity is induced by sub-sampling from a finite dataset. However, heuristics like ADAM may be leveraging less extreme versions of this mechanism. This is promising, as it opens a path towards better algorithms through the development of better estimators in the heavy-tailed setting (Lugosi and Mendelson, 2019; Srinivasan et al., 2021).

However, it is not clear that the proposed mechanism of robust estimation is the cause of this gap in performance. While SGD will not work with an infinite-variance estimator, a heavy-tailed histogram alone, as in Figure 1, is insufficient to infer that SGD will do poorly; there are problems where the histogram appears heavy-tailed but SGD is optimal.[1] Zhang et al. show a link between heavy-tailed stochastic gradient noise and the gap in performance of SGD and heuristics like ADAM and clipping, which we also observe in Figure 2, but the question of whether ADAM outperforms SGD because of this robustness to heavy-tailed stochastic gradient distributions remains open.

We present evidence that stochasticity is not the major factor leading to this performance gap. To control for stochasticity, we show that the performance gap persists or increases as the batch size increases, up to full batch experiments. This indicate that ADAM outperforms gradient descent even without stochasticity, and calls for a rethinking of the mechanism behind its effectiveness.

1. Take the function $f : \mathbb{R}^n \to \mathbb{R}$ defined by $f(x) = \frac{1}{n}\sum_{i=1}^n f_i(x)$ where each f_i depends on a different coordinate, $f_i(x) = \frac{1}{2}(x_i - y_i)^2$. Running SGD with step-size 1 with stochastic gradients ∇f_i corresponds to coordinate descent on $\frac{1}{2}\|x - y\|^2$. This minimizes f after visiting each coordinate, independently of y, while gradient clipping and ADAM fail to converge. By picking y, the histogram of the stochastic gradient error can be made arbitrarily heavy-tailed at initialization, as the stochastic gradients are $[\nabla f_i(x)]_i = x_i - y_i$ and 0 in other coordinates.

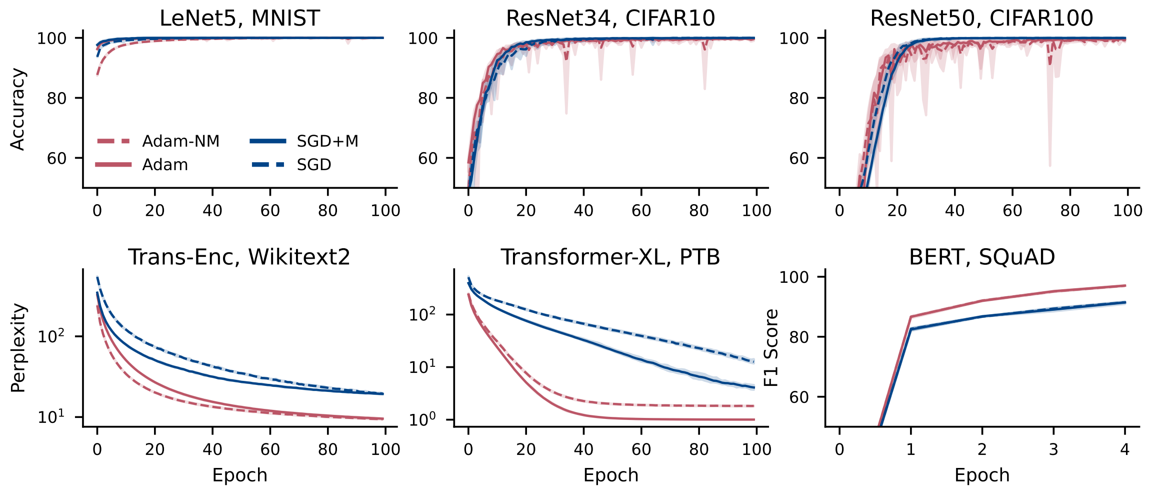

Figure 2: Standard training setup comparing SGD and ADAM with momentum (SGD+M, ADAM) and without (SGD, ADAM-NM). Step-size tuned by grid-search. The performance of SGD and ADAM is barely distinguishable for image problems, while there is a gap on text problems.

2. Experiments controlling for stochasticity

To investigate whether the gap between SGD and ADAM for text data is caused by stochasticity, we select three image problems data as control and three text problems and run the following settings;

(i) training with small batch size to get a baseline of the performance of SGD and ADAM (Figure 2),

(ii) a comparison of the same optimizers as the batch-size increases (Figure 3) and,

(iii) as increasing batch size cannot entirely remove stochasticity for datasets that do not fit in memory, we run the algorithm in full batch on a subset of the dataset (Figure 4).

This last experiment changes the underlying machine learning problem, but the smaller problem should still exhibit similar characteristics in terms of optimization difficulty.

Progress per iteration To check if larger batch sizes lead to more accurate gradient estimates and reduce the gap between SGD and ADAM, we measure progress per iteration rather than per epoch. If stochasticity was not a problem—for the sake of argument, if there was no error in the estimation of the gradient—we would expect no difference in progress per iteration across batch sizes.

Training metrics While the focus in ML is generalization, our focus here is on cases where SGD fails to optimize the training loss in a reasonable time. As many factors beyond the scope of optimization can affect generalization performance, including the heavy-tailedness of stochastic gradients (Şimşekli et al., 2019), we restrict the results to training performance. Figures 2 to 4 present training metrics, while the training loss, which follows similar patterns, is presented in Appendix B.

Accounting for the step-size As increasing the batch size might allow for larger step-sizes to be stable, we tune the step-size by grid-search for all settings. We select the step-size that minimizes the maximum training loss at the end of the training procedure over random initializations.[2] Full experimental details on architectures, datasets and training procedure are deferred to Appendix A.

2. If $l(\alpha, s)$ is the final loss with step-size α and random seed s, we select $\alpha_* = \arg\min_\alpha\{\max_s l(\alpha, s)\}$.

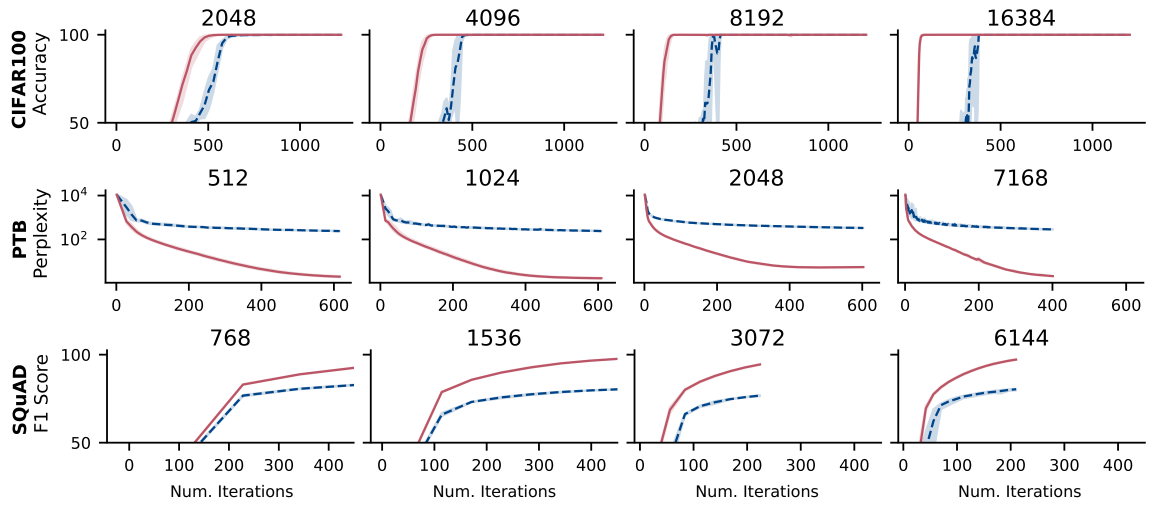

Figure 3: Comparing SGD and ADAM as the batch size increases. When accounting for the selection of the step-size, the gap between SGD and ADAM remains or increases rather than shrink. This pattern holds on the training loss and the other problems, shown in Appendix B.

(i) Standard training Figure 2 shows the performance of training with small batch size, where we observe a larger gap between SGD and ADAM for text data than image data, confirming previous observations. To account for momentum as a confounding factor, we run SGD and ADAM with and without momentum, with default hyperparameter. While the performance depends on momentum for some problems, its influence is much smaller than the gap between algorithms.

(ii) Increasing batch size Figure 3 shows the performance on three problems as the batch size increases. The remaining problems are shown in appendix B. On all problems, the gap in performance at each iteration increases, rather than decrease, with the batch size. While SGD eventually gets the same performance as ADAM for image data, the training curves plateaus for text data. Despite a much larger batch size than in the previous experiment (the smallest batch size in (ii) is between 8 and 32 times that of (i)), the gap in performance between SGD and ADAM does not disappear.

(iii) Deterministic training Figure 4 shows the performance of training in full batch. For the bigger models, the dataset has been subsampled to be computationally feasible. For example, BERT fine-tuning is running in full batch on 7% of the dataset (the training error is also computed on this subset). Due to the smaller dataset for BERT and RESNET50, ADAM achieves high training metrics within the allocated time while SGD stalls, despite stochasticity not being a factor in this setting.

Limitations Due to the increasing batch size, the computational cost of each iteration grows to a point that running the algorithm for many iterations is infeasible given our resources.[3] This approach to control for stochasticity can only probe the "beginning" of the training procedure, in number of parameter updates. As the step-size is selected for the best performance at the end of training, the step-size might also not be stable if the training was allowed to continue.

3. For comparison, the Transformer-Encoder on WIKITEXT2 in experiment (i), with a batch size of 64, goes through 92,800 iterations in ≈1h, while the 1,000 iterations in experiment (iii), with batch size 59,392 (full batch), take ≈6h.

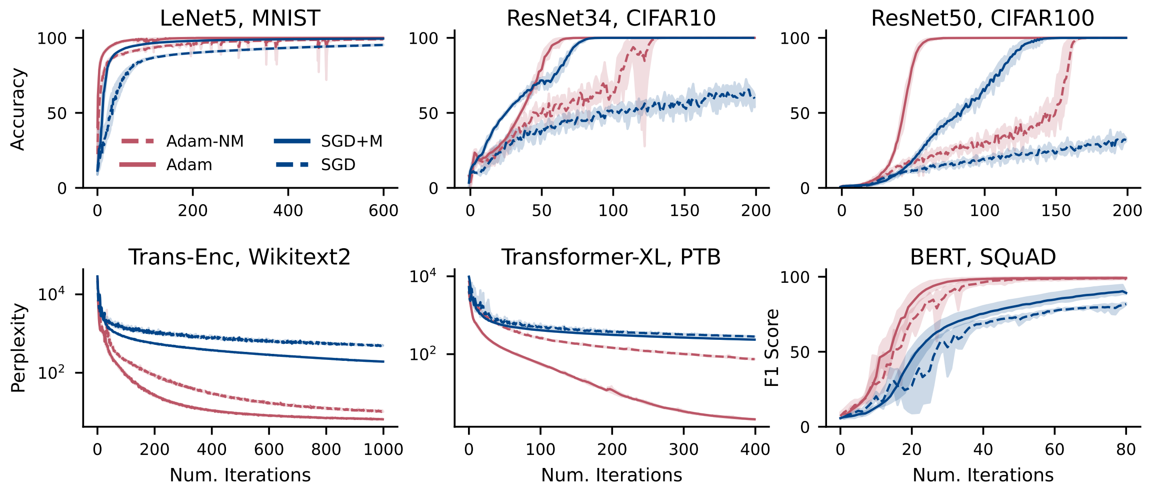

Figure 4: Comparing optimizers running in full batch. For datasets too large to fit in memory, runs are full batch on a subset of the data. Despite having no error in the estimate of the gradient, SGD still lags behind ADAM, and this gap is larger than in the small batch size experiment.

3. Discussion

The experiments indicate that ADAM outperforms SGD regardless of stochasticity. While the heavy-tailed noise hypothesis for the performance of ADAM and clipping can still hold, it does not seem to be the main cause of this gap. This is consistent with previous work on large batch training, such as the work of You et al. (2020) on BERT, which show that ADAM-like methods are still needed in this setting. This suggests that the descent direction used by those methods is better than the gradient.

This raises more questions as to what is the key mechanism that ADAM and other heuristics such as gradient clipping leverage to outperform SGD. A possible explanation, explored in Transformers by Liu et al. (2020), is that ADAM makes the gradient update independent to the magnitude of the gradient along each direction. This matches the view of ADAM as a form of smoothed sign-descent, explored by Balles and Hennig (2018) and the original motivation for RMSPROP (Tieleman and Hinton, 2012). But it is unclear why the gradient magnitude should not be taken into account.

A starting point towards understanding this mechanism would be to simplify ADAM to a simpler core heuristic that performs well in this setting. It seems likely that some of the tricks in ADAM, such as the bias-correction, would have little effect on performance as its effect vanish after a few steps, but other elements are less clear-cut. Is the preconditioning a form of “adaptivity” and a changing learning rate over time, or is it more akin to a sign-descent? Is the element-wise preconditioning necessary, or is it sufficient to scale the update as in ADAGRAD-NORM (Ward et al., 2019)? Is it necessary to smooth the moving average of the preconditioner, or is it sufficient to use only the current gradient?

The success of optimization heuristics in deep learning may not have resulted from optimization performance, but rather from social dynamics or a co-evolution of deep learning optimizers and architectures. But it remains that, in those settings, the heuristics outperform algorithms we understand and can analyze. Understanding the key mechanism behind this success may lead to better algorithms and a better understanding of the problem classes we encounter in deep learning.

References

Alacaoglu, A., Y. Malitsky, P. Mertikopoulos, and V. Cevher (2020). “A new regret analysis for Adam-type algorithms”. In: *ICML 2020*. Vol. 119, pp. 202–210.

Anil, R., V. Gupta, T. Koren, and Y. Singer (2019). “Memory Efficient Adaptive Optimization”. In: *NeurIPS 2019*, pp. 9746–9755.

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). *Layer Normalization*. Preprint. arXiv/1607.06450.

Balles, L. and P. Hennig (2018). “Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients”. In: *ICML 2018*. Vol. 80, pp. 413–422.

Dai, Z., Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov (2019). “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *ACL 2019*, pp. 2978–2988.

Défossez, A., L. Bottou, F. Bach, and N. Usunier (2020). *A Simple Convergence Proof of Adam and Adagrad*. Preprint. arXiv/2003.02395.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL-HLT*, pp. 4171–4186.

He, K., X. Zhang, S. Ren, and J. Sun (2016). “Deep Residual Learning for Image Recognition”. In: *CVPR 2016*, pp. 770–778.

Ioffe, S. and C. Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *ICML 2015*. Vol. 37, pp. 448–456.

Kingma, D. P. and J. Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *ICLR 2015*.

Krizhevsky, A. (May 2012). *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (Dec. 1998). “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86, pp. 2278–2324.

Liu, L., X. Liu, J. Gao, W. Chen, and J. Han (2020). “Understanding the Difficulty of Training Transformers”. In: *EMNLP 2020*, pp. 5747–5763.

Lugosi, G. and S. Mendelson (2019). “Mean Estimation and Regression Under Heavy-Tailed Distributions: A Survey”. In: *Found. Comput. Math.* 19.5, pp. 1145–1190.

Marcus, M. P., B. Santorini, M. A. Marcinkiewicz, and A. Taylor (1999). *Treebank-3*.

Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). “Pointer Sentinel Mixture Models”. In: *ICLR 2017*.

Orabona, F. (2020). *Neural Networks (maybe) Evolved to Make Adam the Best Optimizer*. Blog post. https://parameterfree.com/2020/12/06/.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). “On the difficulty of training recurrent neural networks”. In: *ICML 2013*. Vol. 28, pp. 1310–1318.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *NeurIPS 2019*, pp. 8024–8035.

PyTorch contributors (2021). *Language Modeling with nn.transformer and torchtext*. https://pytorch.org/tutorials/beginner/transformer_tutorial. Accessed: 2021-06-10.

Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). “SQuAD: 100,000+ Questions for Machine Comprehension of Text”. In: *EMNLP 2016*, pp. 2383–2392.

Şimşekli, U., L. Sagun, and M. Gürbüzbalaban (2019). "A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks". In: *ICML 2019*. Vol. 97, pp. 5827–5837.

Srinivasan, V., A. Prasad, S. Balakrishnan, and P. K. Ravikumar (2021). *Efficient Estimators for Heavy-Tailed Machine Learning*. Preprint. openreview:5K8ZG9twKY.

Tieleman, T. and G. Hinton (2012). *RMSPROP: Divide the gradient by a running average of its recent magnitude*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). "Attention is All you Need". In: *NeurIPS 2017*, pp. 5998–6008.

Ward, R., X. Wu, and L. Bottou (2019). "AdaGrad stepsizes: sharp convergence over nonconvex landscapes". In: *ICML 2019*. Vol. 97, pp. 6677–6686.

Wolf, T., 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 (2020). "Transformers: State-of-the-Art Natural Language Processing". In: *EMNLP 2020*, pp. 38–45.

You, Y., J. Li, S. J. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh (2020). "Large Batch Optimization for Deep Learning: Training BERT in 76 minutes". In: *ICLR 2020*.

Zhang, J., S. P. Karimireddy, A. Veit, S. Kim, S. J. Reddi, S. Kumar, and S. Sra (2020). "Why are Adaptive Methods Good for Attention Models?" In: *NeurIPS 2020*.

Supplementary material

Appendix A. Experimental details

Code to reproduce the experiments is available at

github.com/jacqueschen1/adam_sgd_heavy_tails

Experiments are run on a Tesla V100-SXM2 (32GB) or P100-PCIE (12GB) depending on the batch size.

Optimizers We use the PyTorch (Paszke et al., 2019) implementation of

SGD	without momentum
SGD+M	SGD with heavy-ball momentum, fixed at $\beta = 0.9$
Adam	with default hyperparameters ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)
Adam-NM	Adam without momentum ($\beta_1 = 0$).

Grid search The step-size is set by a grid search on each experiment. We run an initial sparse grid and densify it on regions that performed well. Each run is repeated for five random seeds, determining the initialization and the order in which the data is loaded. We select the step-size that minimizes the maximum training loss at the end of the training procedure over random initializations. That is, if $l(\alpha, s)$ is the final loss with step-size α and random seed s, we select $\alpha_* = \arg\min_\alpha\{\max_s l(\alpha, s)\}$. The results of the grid search for each experiment are given in Appendix C.

Datasets and Models We use the following dataset, models and implementations

MNIST, LeNet5 (Lecun et al., 1998)

CIFAR10, ResNet34 (Krizhevsky, 2012; He et al., 2016)
`torchvision` implementation.

CIFAR100, ResNet50 (Krizhevsky, 2012; He et al., 2016)
`torchvision` implementation.

WikiText2, Trans-Enc (Merity et al., 2017; PyTorch contributors, 2021)
Transformer-Encoder from the PyTorch Transformer tutorial, using a target length of 35.

PTB, Transformer-XL (Marcus et al., 1999; Dai et al., 2019)
Implementation of Dai et al. (2019) for the model and dataset preprocessing on PTB/WikiText2. We follow the training procedure of Zhang et al. (2020), reducing the number of layers to 6 and use a target length of 128. Other hyperparameters use the base ENWIK8 experiment from the original Transformer-XL code (but without gradient clipping and learning rate scheduling).
github.com/kimiyoung/Transformer-XL

SQuAD v1.1, BERT (Rajpurkar et al., 2016; Devlin et al., 2019)
Pretrained BERT$_{\text{base}}$ implementation from HuggingFace (Wolf et al., 2020), fine-tuning task on SQuAD using a target length of 384. github.com/huggingface/transformers

A.1. Histogram

The histogram of gradient errors in Figure 1 uses batch size 128 (image data), 64 (Transfomer-Encoder, Transformer-XL) and 24 (BERT). Due to the batch normalization layers (Ioffe and Szegedy, 2015) in ResNet, the concept of a "full" gradient is ill-defined; the average of the gradients of the minibatches is not the gradient of the full dataset passed through the model at once. We use the average over the minibatch gradients as we cannot pass the full dataset through the model at once. Transformer models use layer normalization (Ba et al., 2016) and are unaffected.

A.2. Standard training

The small batch training procedure in Figure 2 uses the same batch size as the histogram experiment; 128 (image data), 64 (Transfomer-Encoder, Transformer-XL) and 24 (BERT).

A.3. Increasing Batch Size

Figure 3 shows the results of the three larger models, RESNET50, Transformer-XL, and BERT. The remaining problems are given in Appendix B. The batch size for each problem is given on the plot. The following table gives a comparison of the largest batch size for each problem and the size of the dataset, showing that the dataset is split in only a few (<15) batches.

Dataset	**Model**	**Dataset Size**	**Largest Batch Size (%)**
MNIST	LENET5	60000	60000 (100%)
CIFAR10	RESNET34	50000	16384 (33%)
CIFAR100	RESNET50	50000	16384 (33%)
WIKITEXT2	TRANS-ENC	59676	59392 (99%)
PTB	Transformer-XL	7263	7168 (99%)
SQUAD v1.1	BERT$_{\text{base}}$	87714	6144 (7%)

For text data, we use gradient accumulation due to limited GPU memory. Incomplete batches at the end of each epoch are dropped to avoid outlier gradients which computed with much less data.

A.4. Deterministic Training

We cap the run-time of all experiments to 6 hours, resulting in the differing number of iterations as Figure 4. For LENET5, TRANS-ENC, and Transformer-XL, we use the same setup as the largest batch size from the increasing batch size as these batch sizes are already full batch. For all settings, the optimization problem is changed to only consider one batch as the full dataset, such that the optimization is deterministic. The reduced dataset sizes are given in the following table.

Dataset	**Model**	**Dataset Size**	**Reduced Dataset Size**
MNIST	LENET5	60000	60000 (100%)
CIFAR10	RESNET34	50000	20000 (40%)
CIFAR100	RESNET50	50000	17000 (34%)
WIKITEXT2	TRANS-ENC	59676	59392 (99%)
PTB	Transformer-XL	7263	7168 (99%)
SQUAD v1.1	BERT$_{\text{base}}$	87714	6144 (7%)

Appendix B. Additional results

Appendix C. Grid-search validation

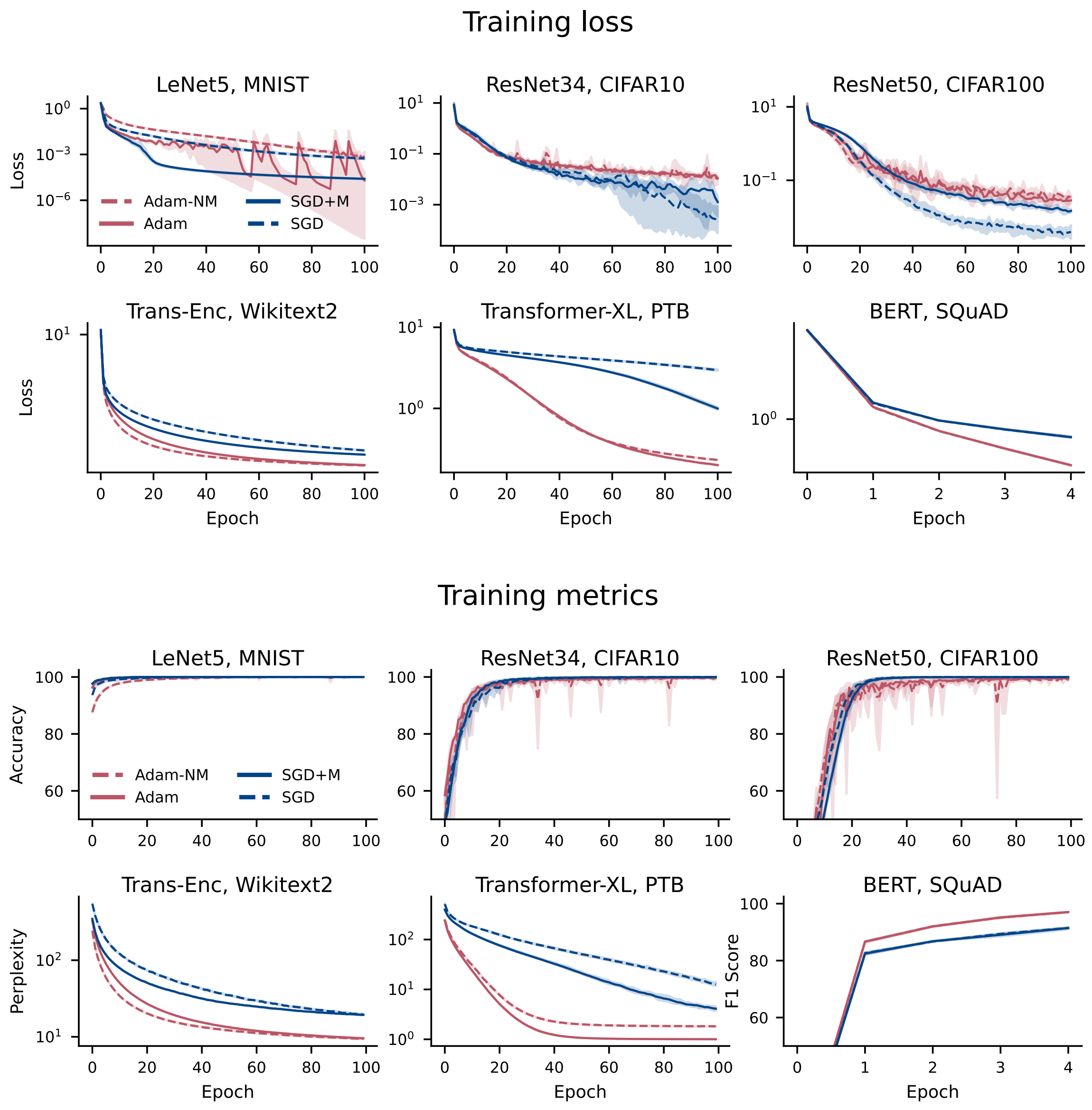

Figure 5: Training loss and metrics for experiment (i), standard training. Training metrics are the same as in Figure 2, reproduced for ease of comparison.

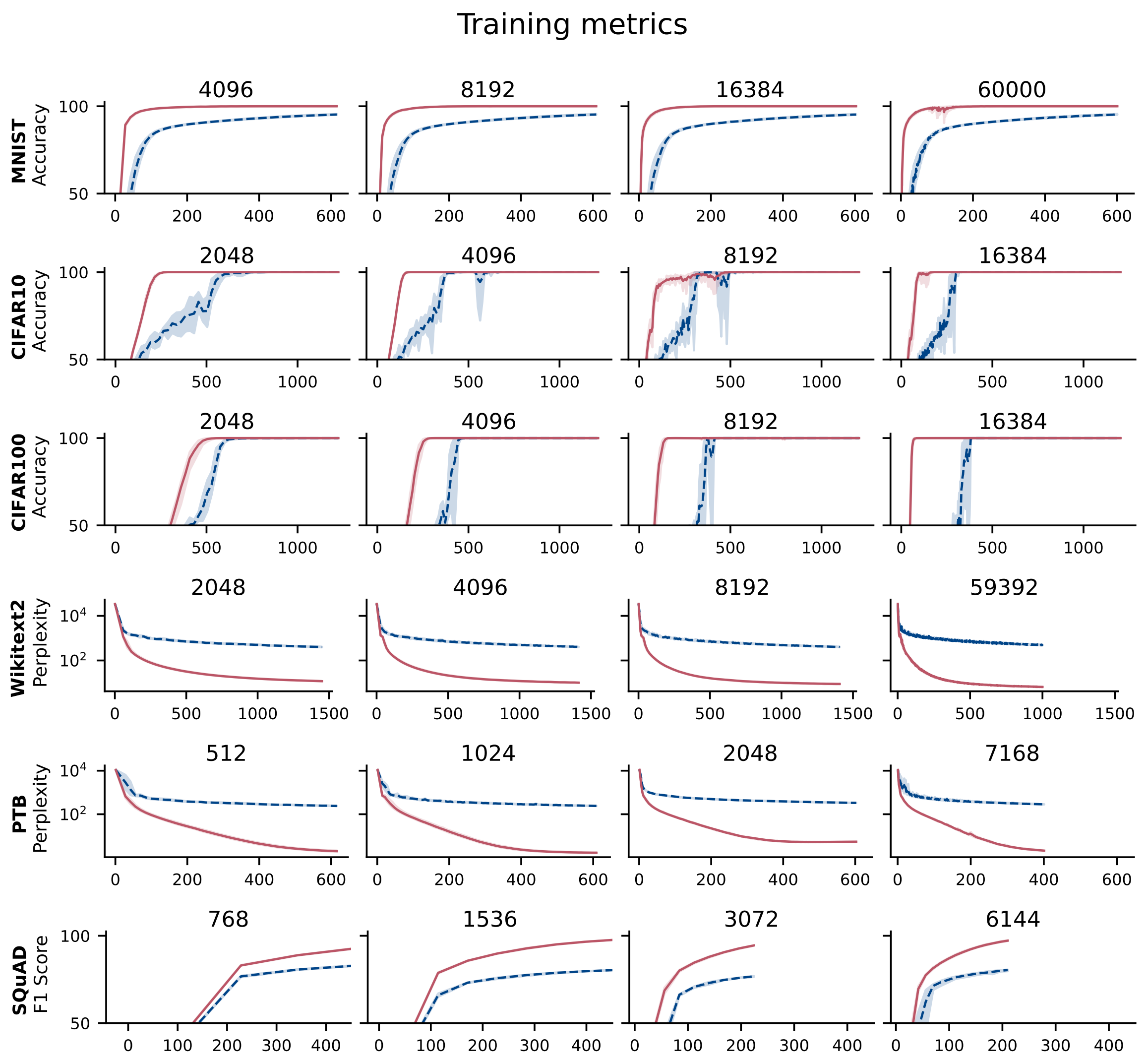

Figure 6: Training metrics for experiment (ii), increasing batch size. Training metrics for CIFAR10/RESNET50, PTB/Transformer-XL, SQUAD/BERT are the same as in Figure 3, reproduced for ease of comparison.

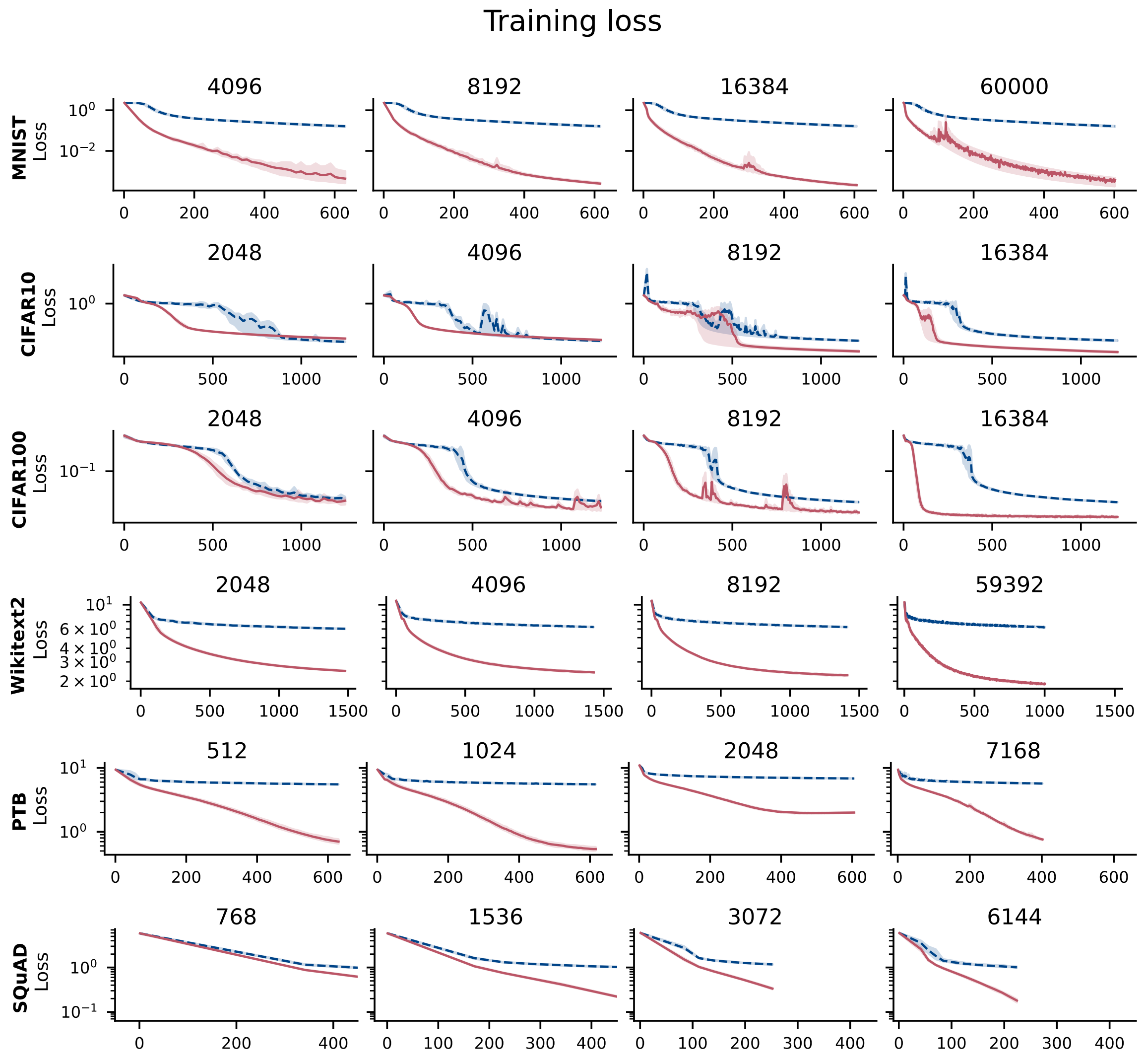

Figure 7: Training loss for experiment (ii), increasing batch size.

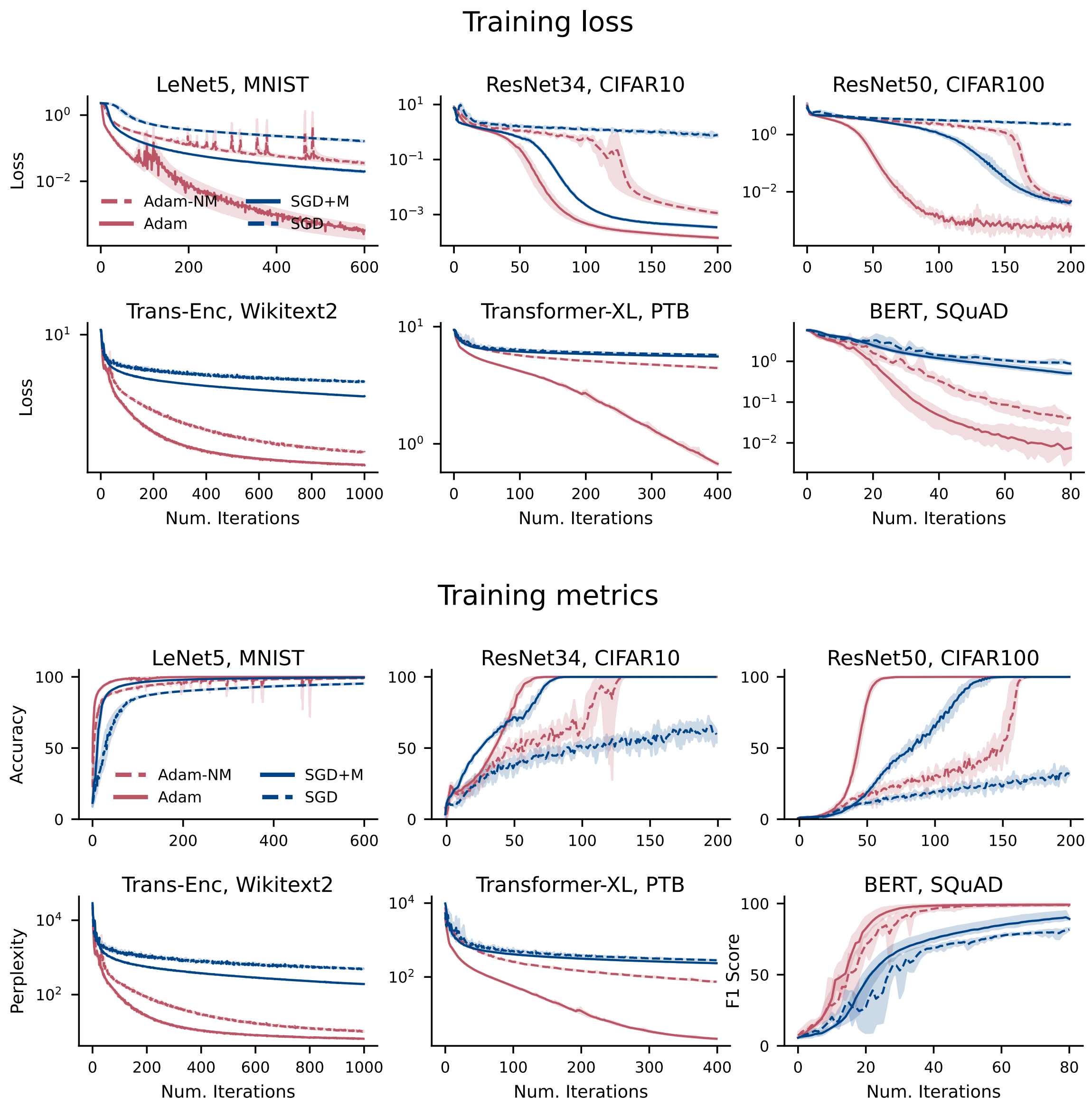

Figure 8: Training loss and metrics for experiment (iii), full batch training. Training metrics are the same as Figure 4, reproduced for ease of comparison.

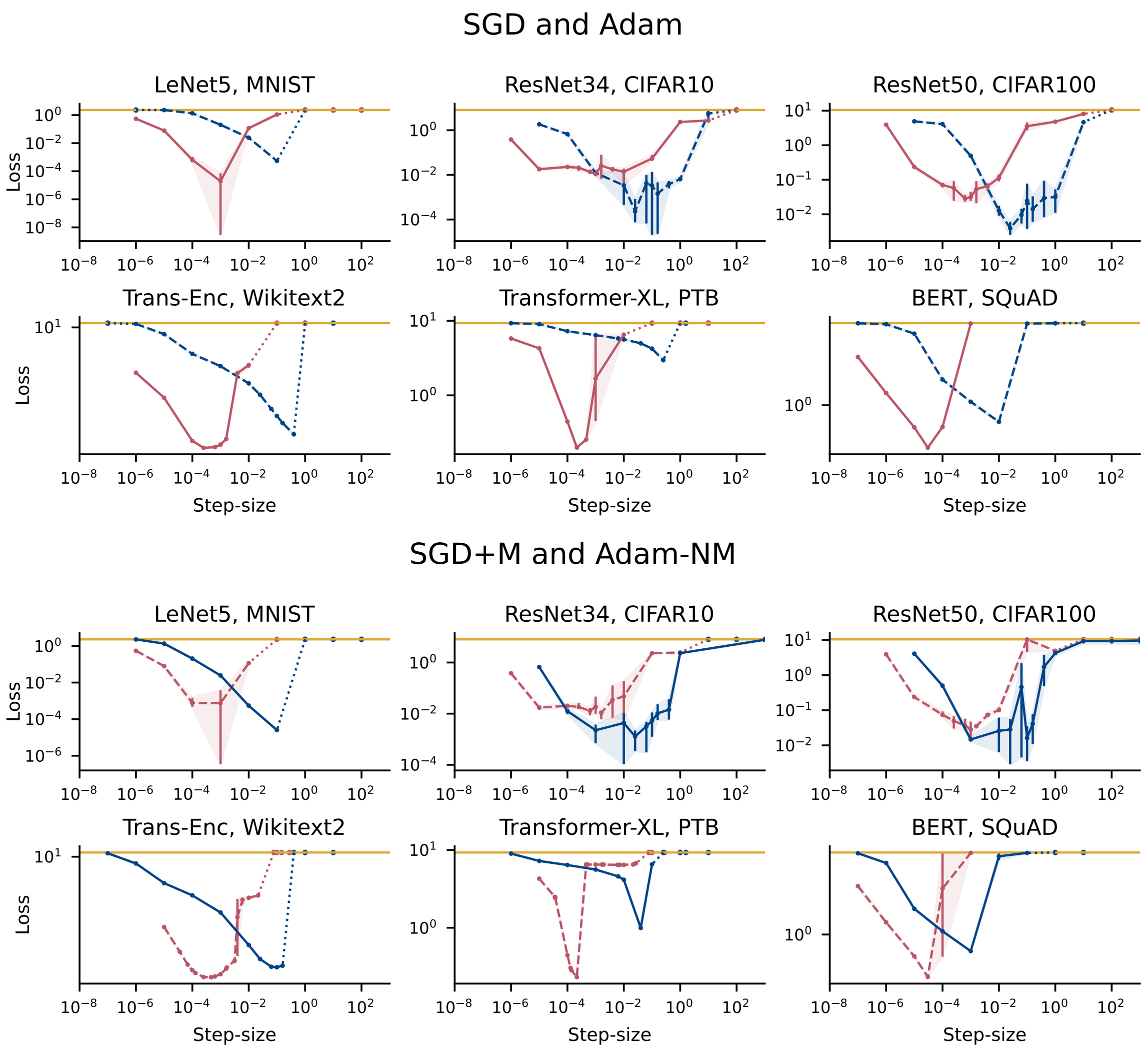

Figure 9: Average final training loss values for every step size in the grid search for experiment (i), standard training. Points on the yellow horizontal bar represent step sizes where the final training loss did not improve from the starting loss or diverged during training. Error bars show the range between the min and max final values among the 5 random seeds.

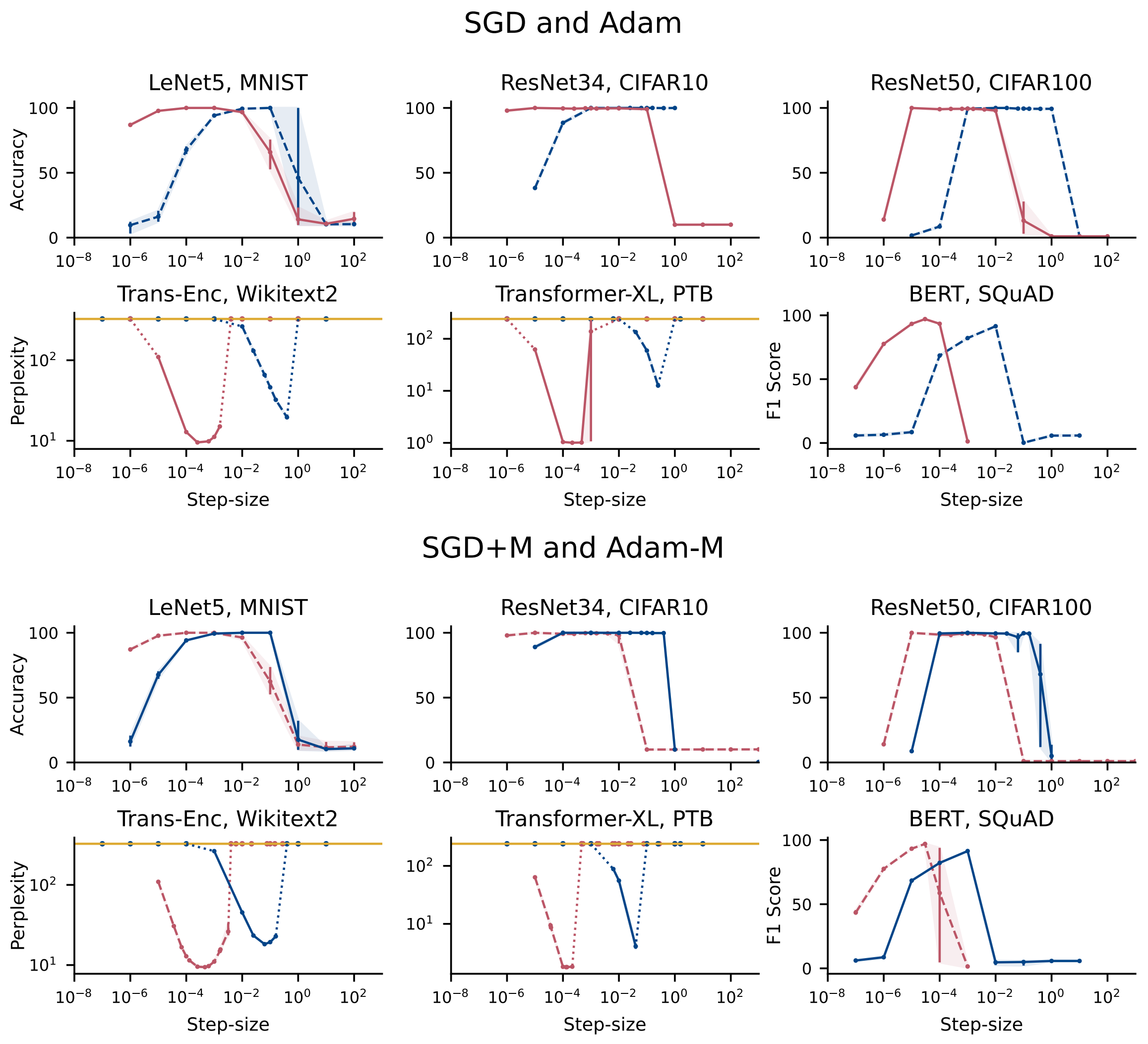

Figure 10: Average final training metric values for every step size in the grid search for experiment (i), standard training. Points on the yellow horizontal bar represent step sizes where the final perplexity did not improve from the start or diverged during training.

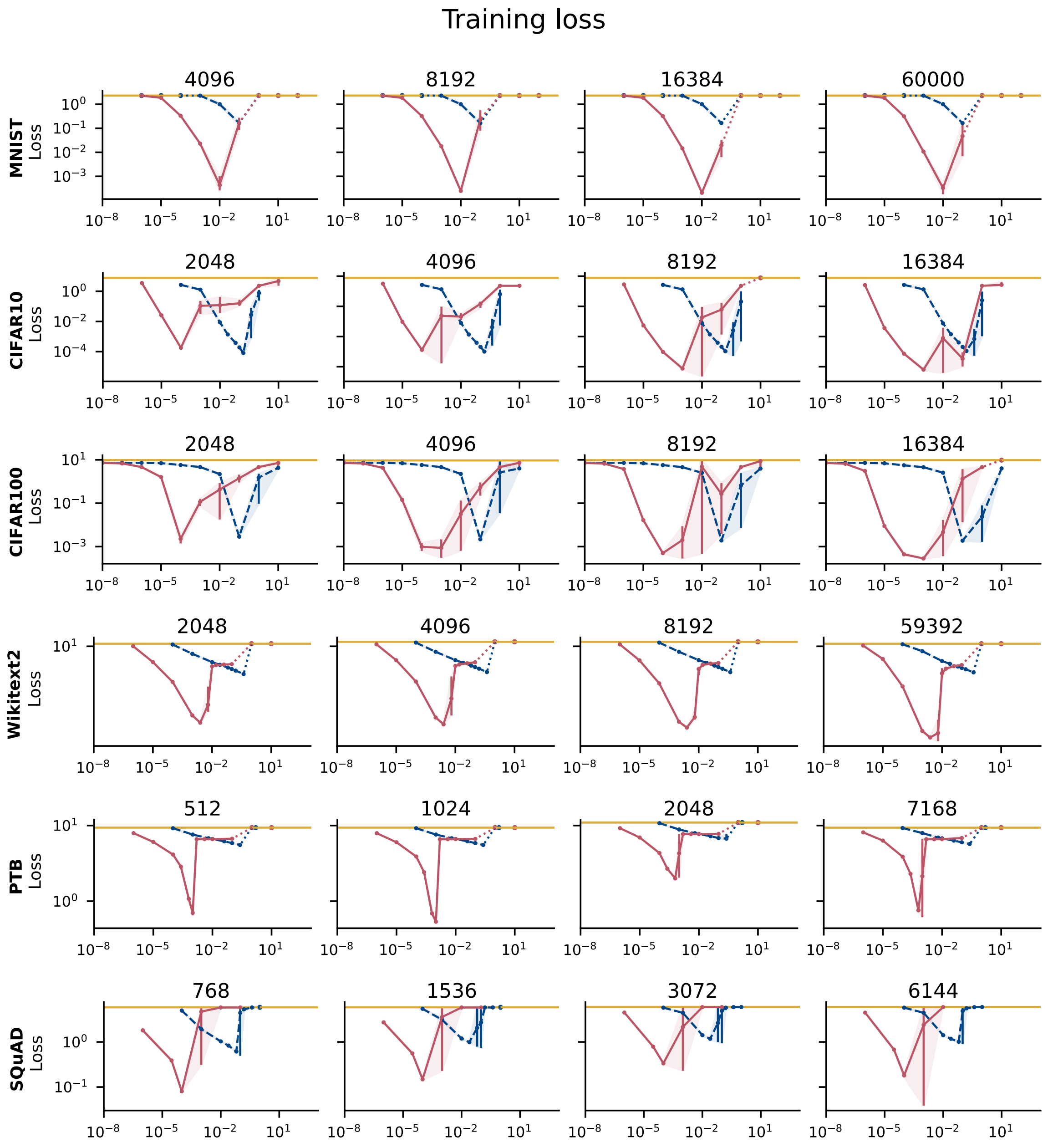

Figure 11: Average final training loss values for every step size in the grid search for experiment (ii), increasing batch size.

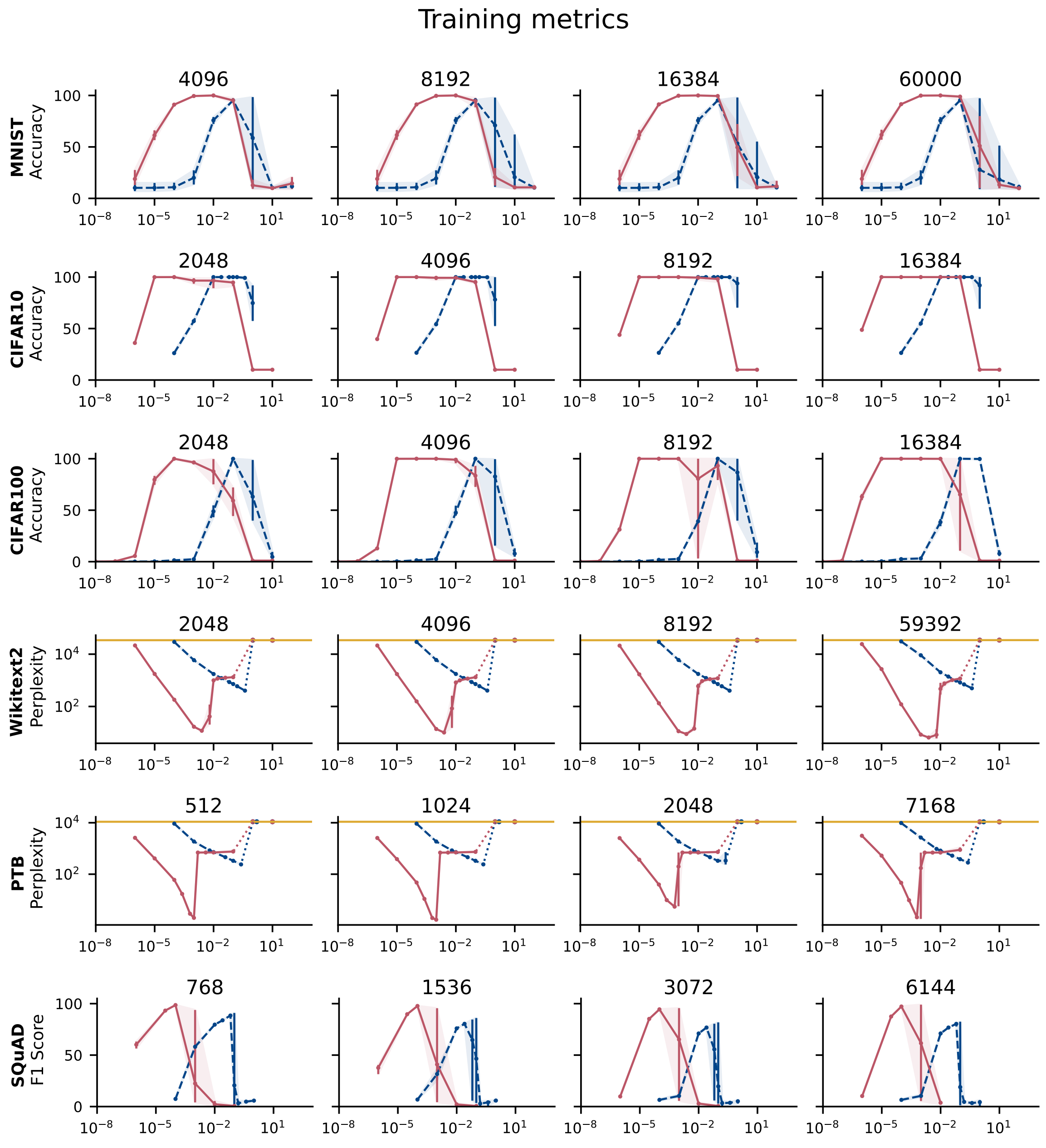

Figure 12: Average final training metric values for every step size in the grid search for experiment (ii), increasing batch size.

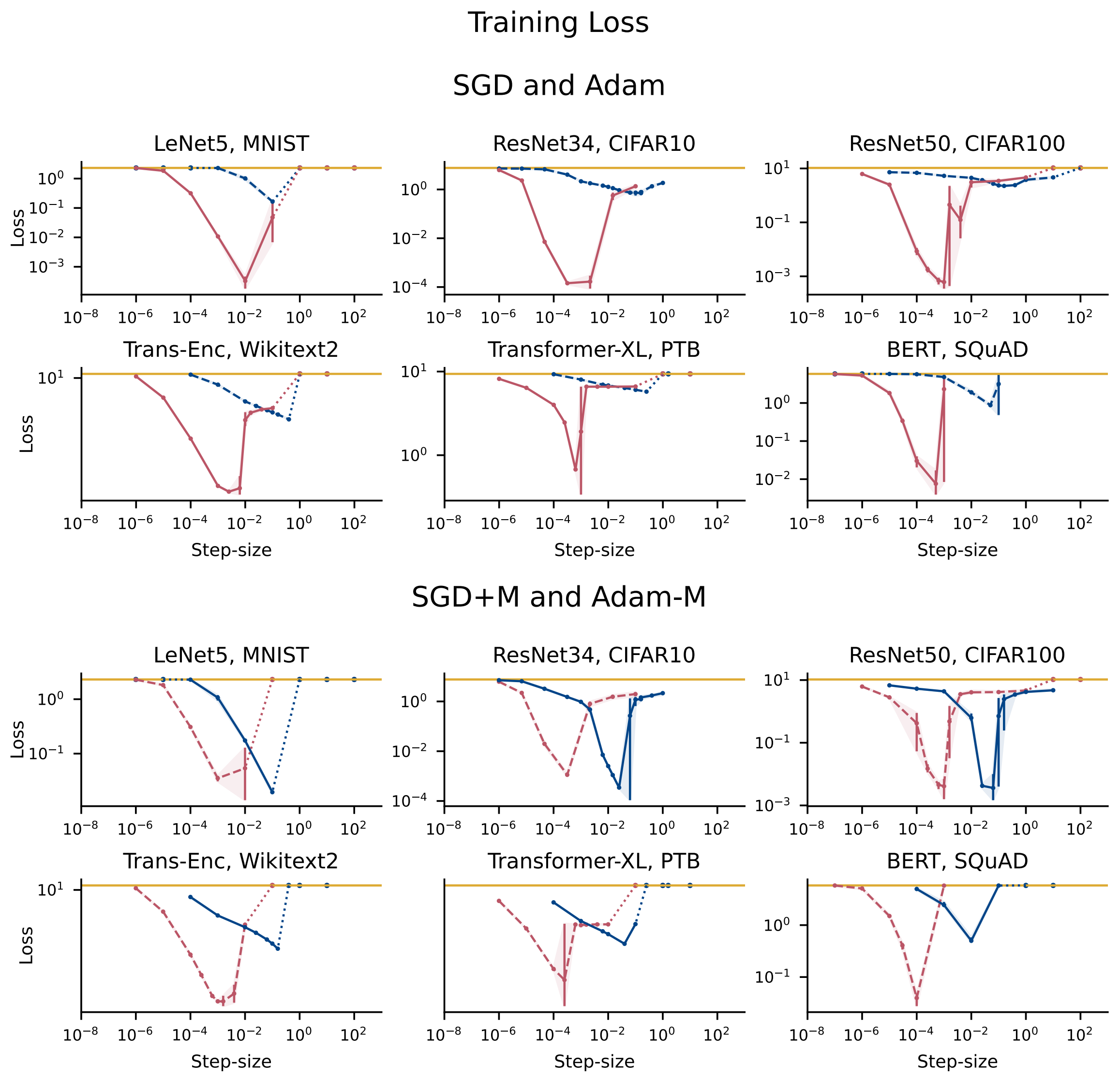

Figure 13: Average final training loss values for every step size in the grid search for experiment (iii), full batch experiments.

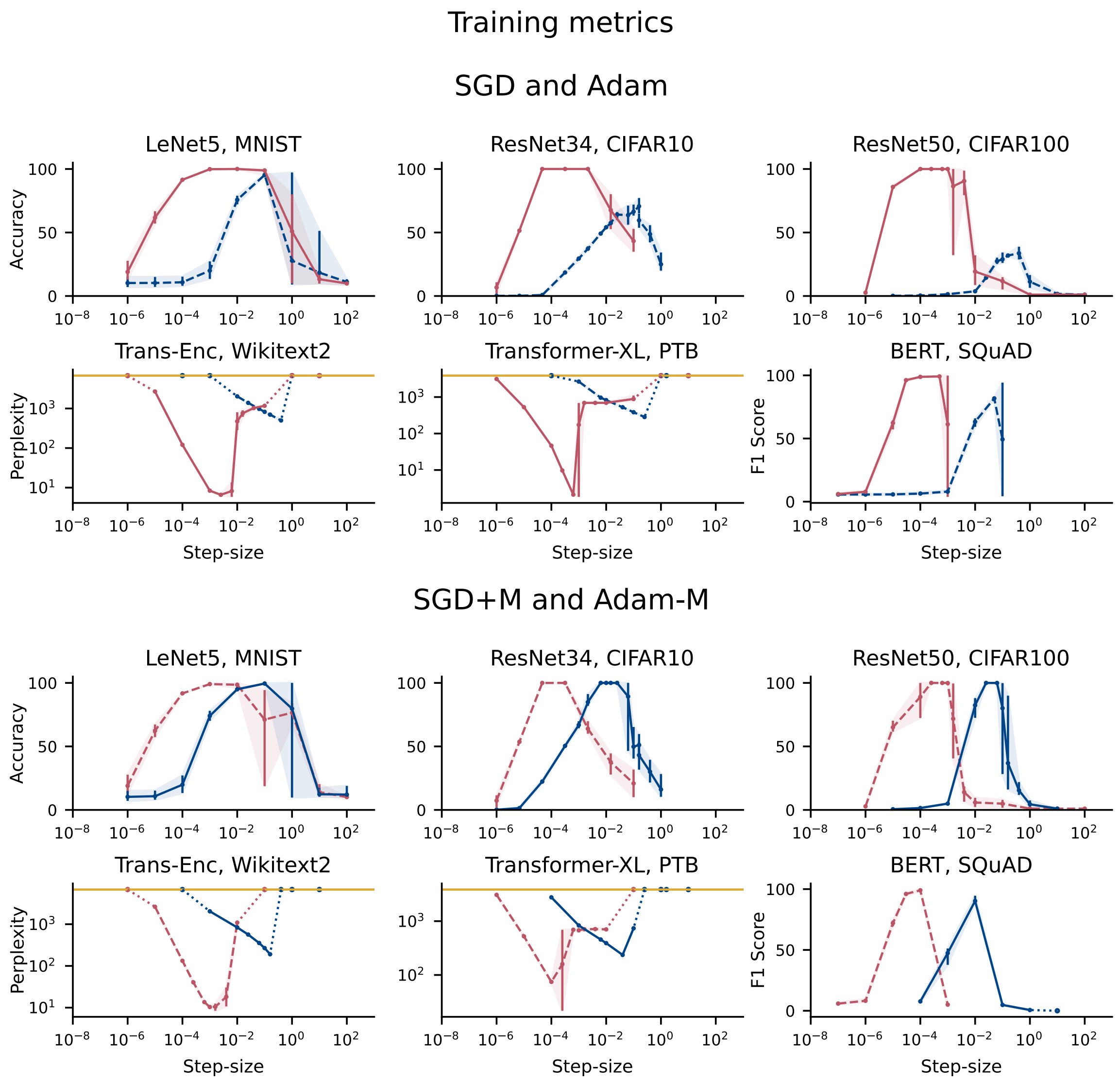

Figure 14: Average final training metric values for every step size in the grid search for experiment (iii), full batch experiments.