Generalization Analysis of Asynchronous SGD Variants

Caspar Amery [*1] **Marco Lourenço** [*1] **Federico Villa** [*1]

Abstract

Asynchronous Stochastic Gradient Descent (ASGD) improves training efficiency by enabling parallel workers to update model parameters asynchronously, which introduces staleness in the updates. While convergence of ASGD variants is well established, their impact on generalization is less explored. Our study shows that these ASGD methods achieve comparable convergence and equal or better generalization than standard SGD despite staleness, with DASGD notably reducing l_2 norm.

1. Introduction

Stochastic Gradient Descent (SGD) is a fundamental optimization technique widely employed in machine learning (ML), typically implemented by sequentially processing mini-batches of data and updating the model parameters. In this standard implementation, a single worker performs computations sequentially, which can become inefficient, especially when dealing with large datasets or models.

To address these efficiency limitations, Asynchronous Stochastic Gradient Descent (ASGD) employs multiple parallel workers that independently compute gradients on separate mini-batches. Unlike synchronous approaches, ASGD applies these gradients to the model parameters immediately upon completion without waiting for other workers to finish. This asynchronous approach introduces "staleness," meaning updates may be based on outdated model parameters due to concurrent modifications by other workers. Although staleness increases stochasticity, it can enhance computational throughput and scalability by effectively leveraging parallel resources.

While previous literature establishes convergence guarantees of various ASGD algorithms, the impact of staleness on model generalization has received limited attention. Given this research gap, our project aims explicitly at evaluating the generalization capabilities of several ASGD algorithms, including **ASAP.SGD** (Bäckström et al., 2022), **SA-ASGD** (Zhang et al., 2016) and **DASGD** (Tan et al., 2024). The theoretical convergence of the examined algorithms is already established in their respective foundational works, allowing us to concentrate exclusively on generalization aspects.

[1]École Polytechnique Fédérale de Lausanne. Correspondence to: Caspar Amery <caspar.amery@epfl.ch>, Marco Lourenço <marco.lourencoleitao@epfl.ch>, Federico Villa <federico.villa@epfl.ch>.

Optimization for machine learning, CS-439, EPFL

2. Model and Dataset

2.1. Dataset

We use synthetic linear regression datasets with three levels of over-parameterization (10%, 50%, 100% of the sample size). Each dataset contains 100 samples with targets generated by a weight vector sampled uniformly from $[-5, 5]$. We split 80%/20% into training/test sets and run 200 independent seeds per configuration. Across all experiments the same batch size (10% of training data) is used.

Learning rate: $\eta_0 = 0.95 \cdot \frac{2}{\sigma^2_{\max}(X)}$, where $\sigma_{\max}$ is the largest singular value of the design matrix X, ensuring stable convergence in our linear over-parametrized problem.

Workers: All ASGD variants use 10 parallel workers in our experiments. Increasing this number empirically changed only the *mean* of the staleness distribution. Thus this should not affect the behavior of our used ASGD algorithms and so we fixed it for all experiments.

2.2. Staleness Distribution Considerations

We hypothesize that the shape of the staleness distribution affects ASGD performance in our tested algorithms and that its impact may vary across algorithm variants. While some theoretical analyses assume simplified models (e.g., uniform or Poisson) for analytical tractability, empirical studies show that staleness distributions are highly context-dependent and influenced by the specific AsyncSGD algorithm, number of workers, and system-level factors such as hardware heterogeneity and consistency mechanisms. Observed distributions can exhibit diverse forms including multi-modal or normal like patterns depending on worker update rates and system conditions. Although the expected staleness increases roughly linearly with the number of workers, there is no single canonical parametric form such as exponential that consistently fits real scenarios.

To maintain transparency, we report our observed staleness histograms per experiment in the appendix.

2.3. Evaluation Metrics

We use several complementary metrics to quantify the generalization and structural properties of the learned models:

Metric	Formula	Intuition and Relevance
Test loss	$\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$	Standard prediction error on unseen test data.
L2 norm	$\|\theta\|_2=\sqrt{\sum_i \theta_i^2}$	Measures model complexity; lower values typically correlate with better generalization.
Sparsity ratio	$\frac{\#(\theta_i\approx 0)}{d}$	Indicates how many features are effectively ignored; promotes simpler models.
Kurtosis	$\frac{1}{3n}\sum_i\left(\frac{\theta_i-\mu}{\sigma}\right)^4$	Detects heavy tails and outlier dominance; large kurtosis may signal overfitting.

Additional computations: **Paired t-tests** for statistical significance. These tools provide a comprehensive view into generalization beyond test loss alone. For each run training stops after reaching approximate 0 training loss.

3. Experiments

3.1. ASAP SGD

ASAP.SGD (Bäckström et al., 2022) uses an adaptive learning rate based on the staleness distribution of updates to mitigate the adverse effects of stale gradients while preserving convergence guarantees. The method uses a **staleness-adaptive step size** $\eta(\tau:\eta_0)$, where τ is the staleness of an update and η_0 is the base learning rate. The function must satisfy two key properties:

- **Mean-preserving:** $\mathbb{E}[\eta(\tau:\eta_0)]=\eta_0$
- **Priority-preserving:** $\eta(\tau+1:\eta_0)\leq\eta(\tau:\eta_0)$

TAIL-τ step size. A step size function satisfying these properties is the TAIL-τ function:

$$\eta(\tau:\eta_0)=C_A(\tau)\,\eta_0,\quad C_A(\tau)=1+A\,(1-2F_{\tilde{\tau}}(\tau))\,,$$

where $F_{\tilde{\tau}}(\tau)=\Pr[\tilde{\tau}\leq\tau]$ is the empirical cumulative distribution function (CDF) of the observed staleness, and A is the amplitude parameter specifying the maximum deviation of η from the base step size η^0.

Theoretical properties. Using this TAIL-τ function additionally guarantees:

- **Range bounds:** $\max_\tau\eta(\tau)=(1+A)\eta_0$ $\min_\tau\eta(\tau)=(1-A)\eta_0$
- **Variance:** $\mathrm{Var}[\eta(\tau)]=\frac{(A\eta_0)^2}{3}$

Execution model assumptions. ASAP.SGD assumes: $\mathbb{E}[\tau_t]=\bar{\tau}\ \ \forall t$, and $\mathbb{E}[\tau_t|\tau_{t'}]=\mathbb{E}[\tau_t]\ \ \forall t<t'$

Convergence conditions. ASAP.SGD establishes convergence under the following assumptions: **Lipschitz gradients** (see Equation (1)) and **Bounded gradient moment** (see Equation (2)).

Practical usage. The ASAP.SGD paper (Bäckström et al., 2022) demonstrates that the TAIL-τ step size improves convergence speed in wall-clock time across diverse models and datasets. However, it does not evaluate the effect on generalization, a gap we aim to address in this work. Our experimental setup satisfies all theoretical conditions for convergence, including non-anticipative staleness and bounded gradient assumptions. We follow the original paper in fixing $A=1$. To ensure stability, we scale the base learning rate η_0 by 0.5, so the maximum adaptive learning rate $(1+A)\eta_0$ matches the original baseline η_0 that guarantees stable convergence and is used in our regular SGD implementation.

3.2. Dynamic Async-SGD

DASGD (Tan et al., 2024) employs an adaptive learning rate based on update staleness, mitigating the impact of outdated gradients while ensuring convergence. It updates model parameters using the staleness τ and the number of workers $\mathcal{K}$, defined as follows:

Definition. The update rule is defined as:

$$\begin{cases}\Delta W_{t+1}=\dfrac{\tau}{\tau+\mathcal{K}}\,\Delta W_t+\dfrac{\mathcal{K}}{\tau+\mathcal{K}}\,(-\eta_0\nabla L(W_{t-\tau}))\\ W_{t+1}=W_t+\Delta W_{t+1}\end{cases}$$

Where W_t is the model's weight at time step t and $\nabla L(W_{t-\tau})$ is the gradient computed by a worker with staleness τ.

Interpretation. In this rule, the gradient update is a weighted average of two components:

- The previous update ΔW_t, weighted by the normalized staleness $\frac{\tau}{\tau+\mathcal{K}}$,
- The new gradient, weighted by the normalized number of workers $\frac{\mathcal{K}}{\tau+\mathcal{K}}$.

This weighting scheme balances the influence of past updates and incoming gradients. Since the weights lie $\in[0,1]$ and sum to 1, the update remains stable even with delayed gradients. Higher staleness increases reliance on past updates, while lower staleness gives more weight to the fresher, and typically more relevant, gradient information.

Convergence conditions. The authors showed DASGD convergence under two mild assumptions, which are satisfied in our setting: **Lipschitz gradients** (see Equation (1)) and μ**-strong convexity** (see Equation (3)).

3.3. Staleness-Aware Async-SGD

SA-ASGD (Zhang et al., 2016) uses a staleness-adaptive learning rate and a gradient accumulator to mitigate the impacts of delayed gradient updates in asynchronous training.

The algorithm scales gradients inversely to the workers' staleness τ and accumulates received updates until a threshold c is reached, upon which the received c updates are all applied at the same time to the global model the parameter server stores. This algorithm reduces the effect the staleness has on the overall updates.

Staleness-Adaptive Learning Rate: the staleness is defined as $\tau = t - w$ for a worker computing a model updated on a local version w and the current server having a global model at version t. Gradients are scaled by a variable learning rate $\eta(\tau)$ defined as follows:

$$\eta(\tau) = \begin{cases} \alpha_0/\tau & \text{if } \tau > 0, \\ \alpha_0 & \text{otherwise.} \end{cases}$$

Where α_0 is the base learning rate. The algorithm uses staleness to penalize updates by making them less effective.

Gradient Accumulator: Scaled gradients are saved into a buffer $G = \{\eta(\tau_m) \cdot g_m, ..., \}$ and applied to the parameter server's global model only after they exceed a certain threshold $c = \max(1, \lfloor \text{num\_workers}/\text{max\_staleness} \rfloor$.

Update Rule: The algorithm first computes the accumulated scaled gradient effect $\theta_{t+1} = \theta_t - g_i$, and then updates the global model as $g_i = \frac{1}{c} \sum_{l=1}^{c} \eta(\tau_{i,l}) \cdot g_l$.

Convergence Conditions: in (Zhang et al., 2016) it is demonstrated that under the **Lipschitz gradients** (see Equation (1)) and **Bounded Staleness** (see Equation (4)) conditions and the learning rate constraints (see Equations (5) and (6)), SA-ASGD converges.

4. Results

We moved all tables, raw numerical results and plots to the Appendix to keep the main text concise and focused on the interpretation of results. The experimental results indicate that ASAP.SGD and DASGD generalizes similarly to traditional sequential SGD across various levels of over parameterization. Minimal and inconsistent statistical differences across metrics suggest robustness to staleness-induced stochasticity. Minor significant effects, such as kurtosis in the lowest over parameterization scenario, might reflect subtle differences in weight distributions due to asynchronous updates.

These outcomes highlight that ASGD can exploit parallel computing effectively without significantly sacrificing generalization performance, supporting its practical application in large-scale machine learning tasks.

We observe a highly significant difference between DASGD and SGD on the l_2 norm, with a moderate effect size (Cohen's $d = 0.478$), which remains significant after Bonferroni correction for multiple comparisons. However, unlike the original paper, we did not observe any performance differences in our setup when varying the number of workers.

5. Conclusion

Our project's primary aim was to investigate the generalization performance of three variants of ASGD. Our findings are particularly encouraging because they indicate that these ASGD variants, which were designed to converge faster through parallelization (as shown in their respective papers), do not suffer or decrease in generalization performance compared to regular SGD despite the staleness. This means that not only can these ASGD variants accelerate training with multiple workers, but they can also achieve the same generalization performance as regular SGD, making them practical and effective options for large-scale machine learning tasks. Note that, DASGD reduced l_2 norm suggesting that this model may provide better generalization than standard SGD not only in our simple linear setting but maybe also in other contexts.

Further research should investigate a broader range of ML models (such as deep neural networks and convolutional architectures) and real-world datasets to assess whether our findings generalize beyond simple linear (over parametrized) regression tasks. Additional work could also examine how ASGD staleness interacts with complex data distributions, task complexity, and advanced regularization techniques. Moreover, exploring the impact of different staleness distributions, worker heterogeneity, and implementation details would provide a more complete understanding of generalization under realistic training conditions.

References

Bäckström, K., Papatriantafilou, M., and Tsigas, P. ASAP.SGD: Instance-based adaptiveness to staleness in asynchronous SGD. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 1261–1276. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/backstrom22a.html.

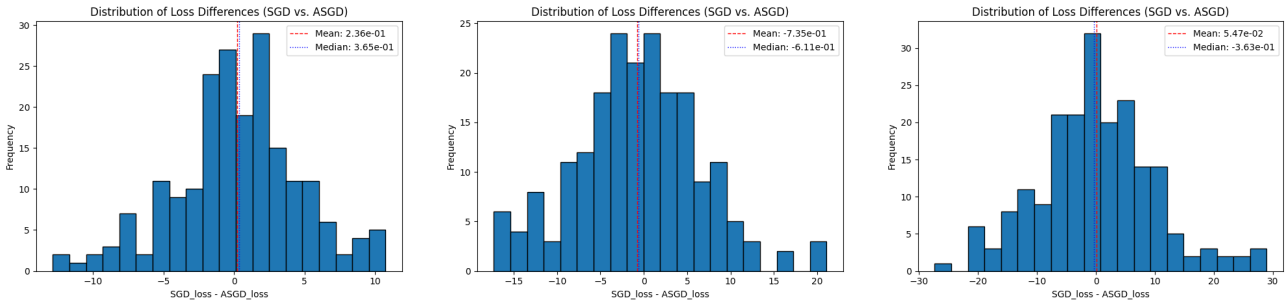
Tan, T., Xie, H., Xia, Y., Shi, X., and Shang, M. Asynchronous sgd with stale gradient dynamic adjustment for deep learning training. *Inf. Sci.*, 681(C), October 2024. ISSN 0020-0255. doi: 10.1016/j.ins.2024.121220. URL https://doi.org/10.1016/j.ins.2024.121220.

Zhang, W., Gupta, S., Lian, X., and Liu, J. Staleness-aware async-sgd for distributed deep learning, 2016. URL https://arxiv.org/abs/1511.05950.

A. Complete Experimental Results ASAP vs SGD

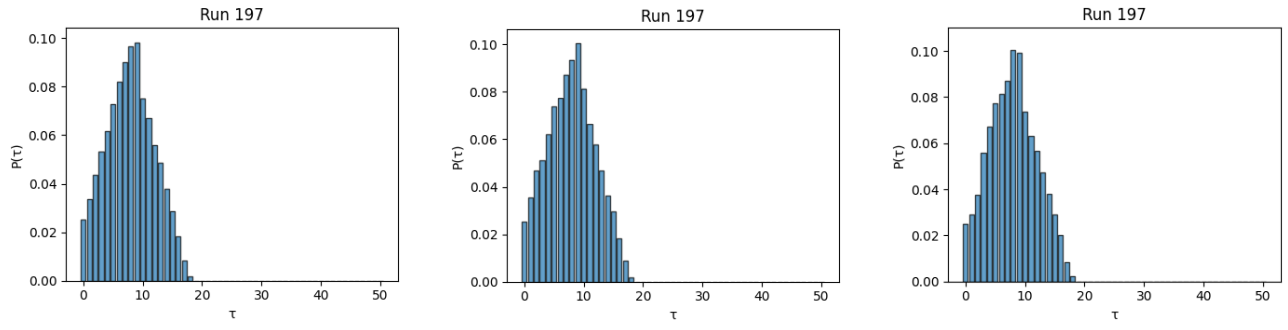
Overparam.	Avg. ASAP Loss	Loss Mean Diff.	Loss diff t-test p	L2 Diff	L2 p	Sparsity Diff	Sparsity p	Kurtosis Diff	Kurtosis p
10%	92.26	0.24	0.450	0.0000	0.802	-0.0003	0.889	-0.0040	0.030
50%	204.32	-0.73	0.156	0.0002	0.013	0.0029	0.273	0.0011	0.733
100%	348.25	0.05	0.938	0.0000	0.802	0.0008	0.663	0.0058	0.030

Table 1. Summary of ASGD vs. SGD experiments across different over parameterization levels for 200 different seeds/runs. Metrics reported as SGD minus ASGD.



(a) 10% overparam. (b) 50% overparam. (c) 100% overparam.

Figure 1. Comparison of test-difference distributions



(a) 10% overparam. (b) 50% overparam. (c) 100% overparam.

Figure 2. Example staleness distribution for each case of over parametrization

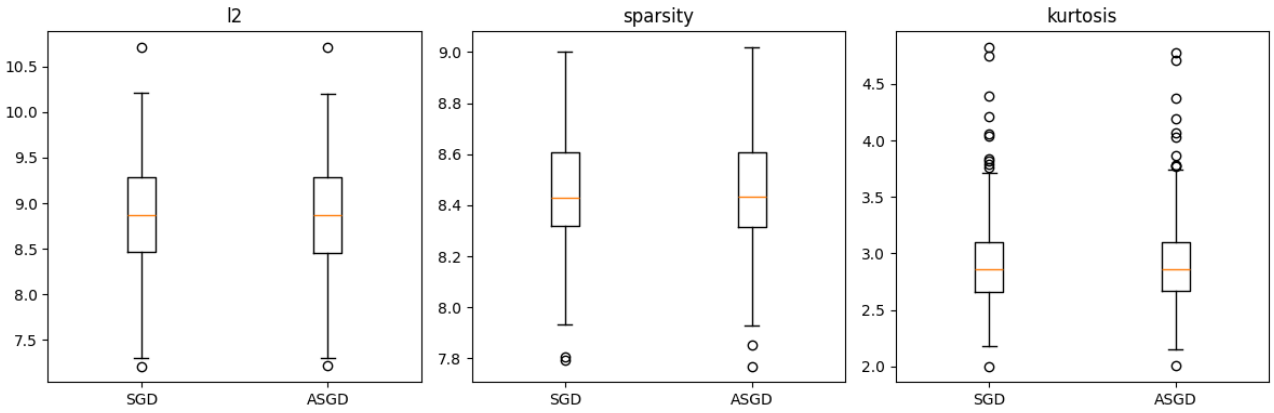


Figure 3. Box plots of per-run weight statistics for SGD vs. ASGD under 10% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

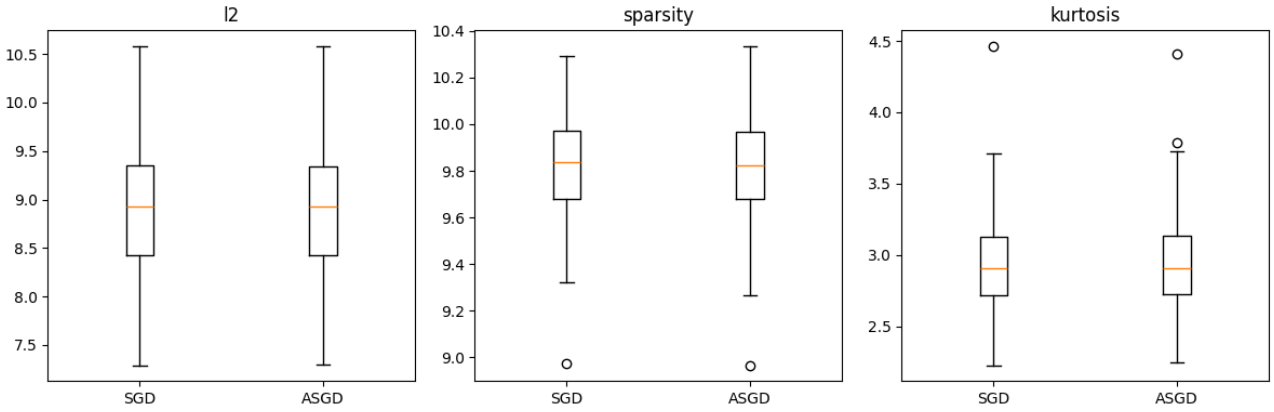


Figure 4. Box plots of per-run weight statistics for SGD vs. ASGD under 50% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

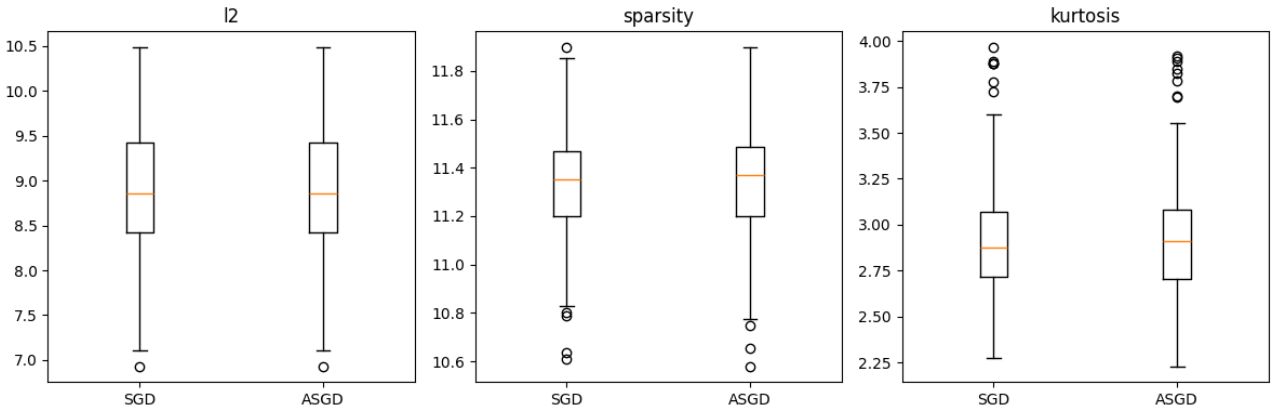
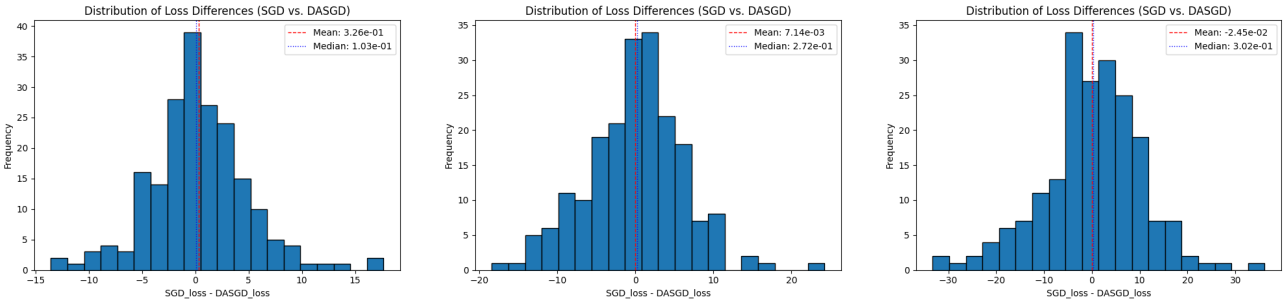


Figure 5. Box plots of per-run weight statistics for SGD vs. ASGD under 100% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

B. Complete Experimental Results DASGD vs SGD

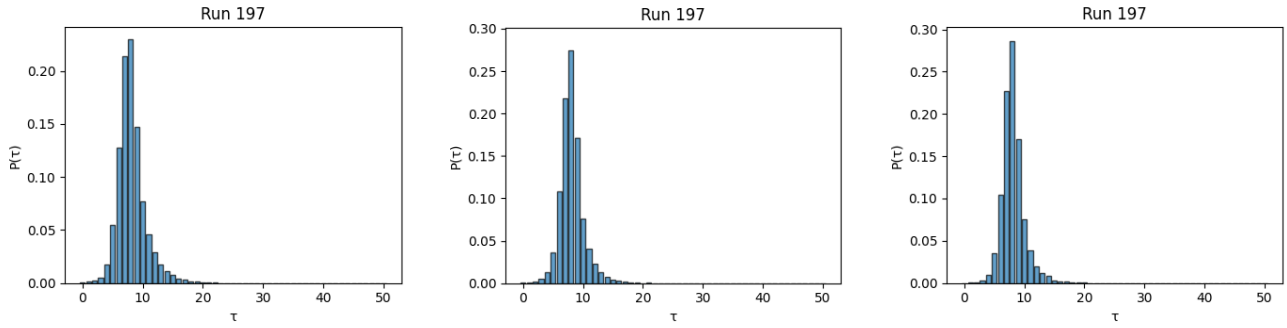
Overparam.	Avg. DASGD Loss	Loss Mean Diff.	Loss diff t-test p	L2 Diff	L2 p	Sparsity Diff	Sparsity p	Kurtosis Diff	Kurtosis p
10%	92.37	0.330	0.320	**0.001**	**0.000**	0.003	0.205	0.0020	0.609
50%	204.07	0.007	0.990	0.000	0.684	0.004	0.137	-0.0040	0.234
100%	348.91	-0.025	0.974	-0.000	0.194	0.002	0.555	0.0007	0.824

Table 2. Summary of DASGD vs. SGD experiments across different over parameterization levels for 200 different seeds/runs. Metrics reported as SGD minus DASGD.



(a) 10% overparam. (b) 50% overparam. (c) 100% overparam.

Figure 6. Comparison of test-difference distributions



(a) 10% overparam. (b) 50% overparam. (c) 100% overparam.

Figure 7. Example staleness distribution for each case of over parametrization

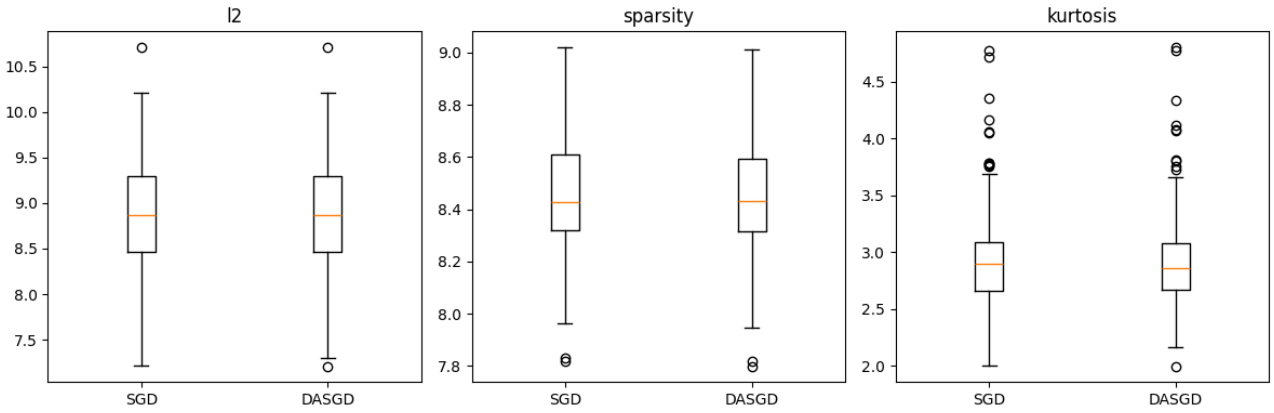


Figure 8. Box plots of per-run weight statistics for SGD vs. DASGD under 10% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

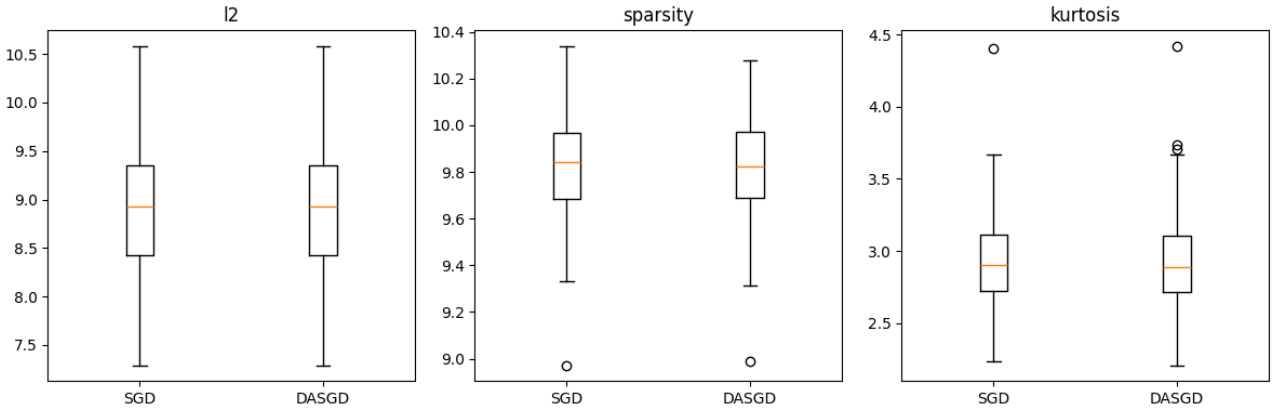


Figure 9. Box plots of per-run weight statistics for SGD vs. DASGD under 50% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

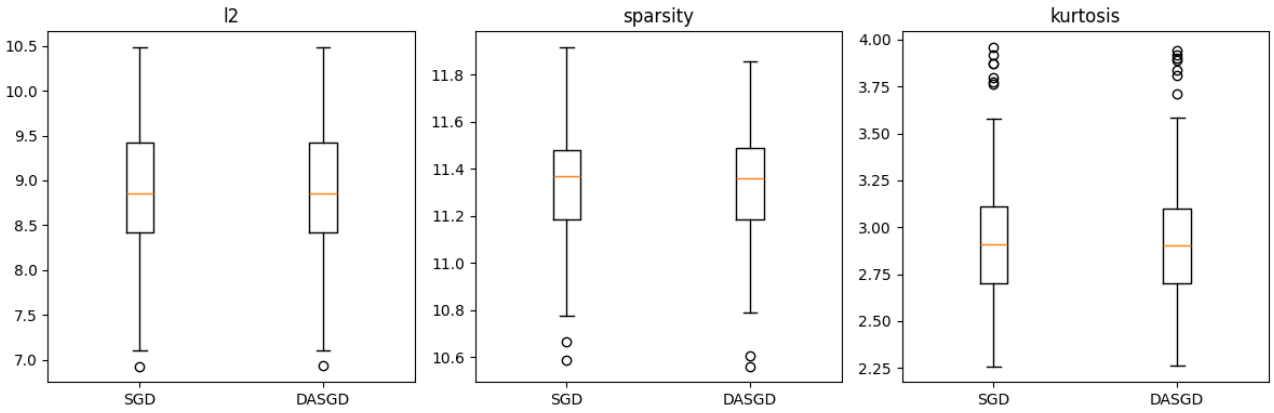
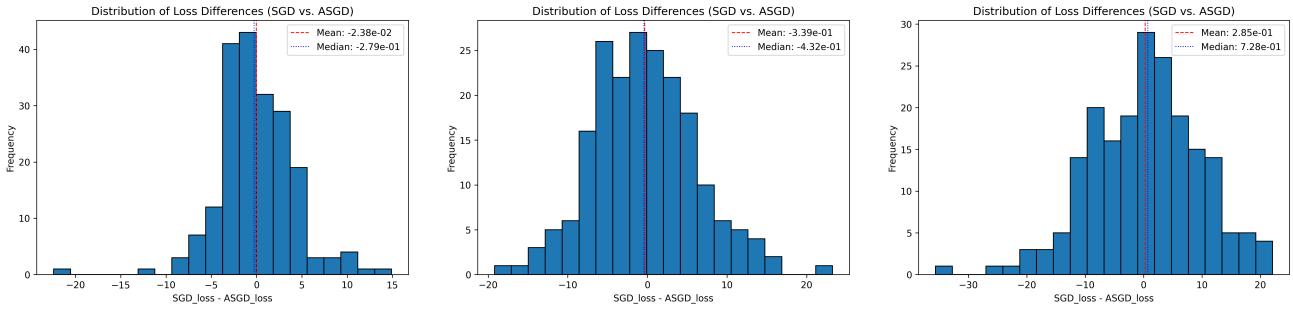


Figure 10. Box plots of per-run weight statistics for SGD vs. DASGD under 100% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

C. Complete Experimental Results SA-ASGD vs SGD

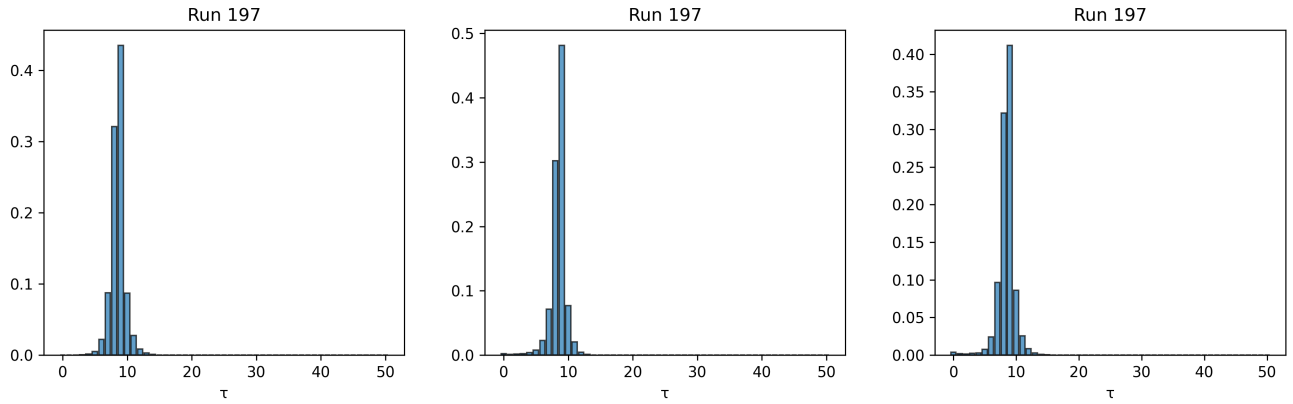
Overparam.	Avg. SA-ASGD Loss	Loss Mean Diff.	Loss diff t-test p	L2 Diff	L2 p	Sparsity Diff	Sparsity p	Kurtosis Diff	Kurtosis p
10%	92.52	-0.0237	0.937	0.0014	0.000	0.0014	0.539	-0.0031	0.2798
50%	203.93	-0.3393	0.466	0.0001	0.3728	-0.0017	0.5320	0.0052	0.1420
100%	348.642	0.2854	0.6713	0.000	0.7326	0.0017	0.5928	0.0010	0.7628

Table 3. Summary of SA-ASGD vs. SGD experiments across different over parameterization levels for 200 different seeds/runs. Metrics reported as SGD minus SA-ASGD.



(a) 10% overparam. (b) 50% overparam. (c) 100% overparam.

Figure 11. Comparison of test-difference distributions



(a) 10% overparam. (b) 50% overparam. (c) 100% overparam.

Figure 12. Example staleness distribution for each case of over parametrization

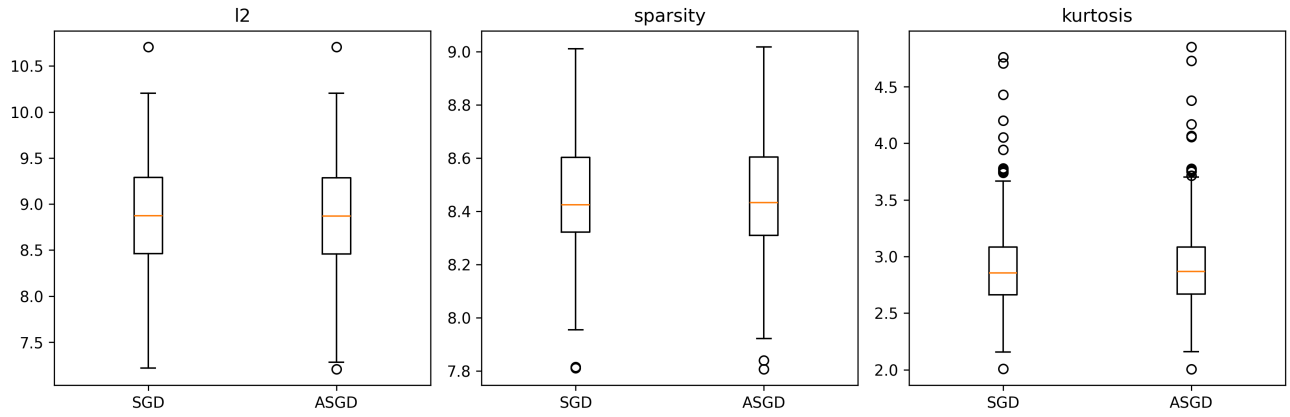


Figure 13. Box plots of per-run weight statistics for SGD vs. SA-ASGD under 10% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

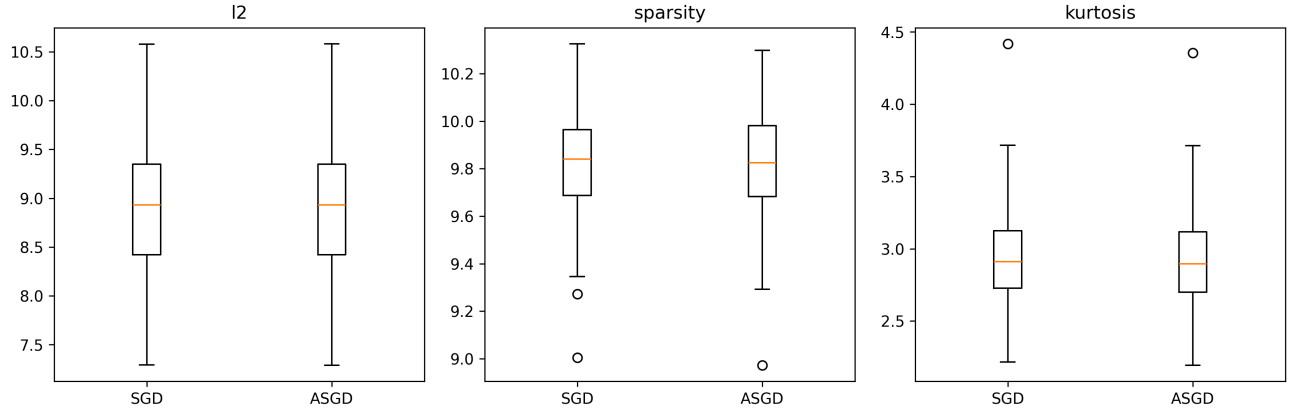


Figure 14. Box plots of per-run weight statistics for SGD vs. SA-ASGD under 50% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

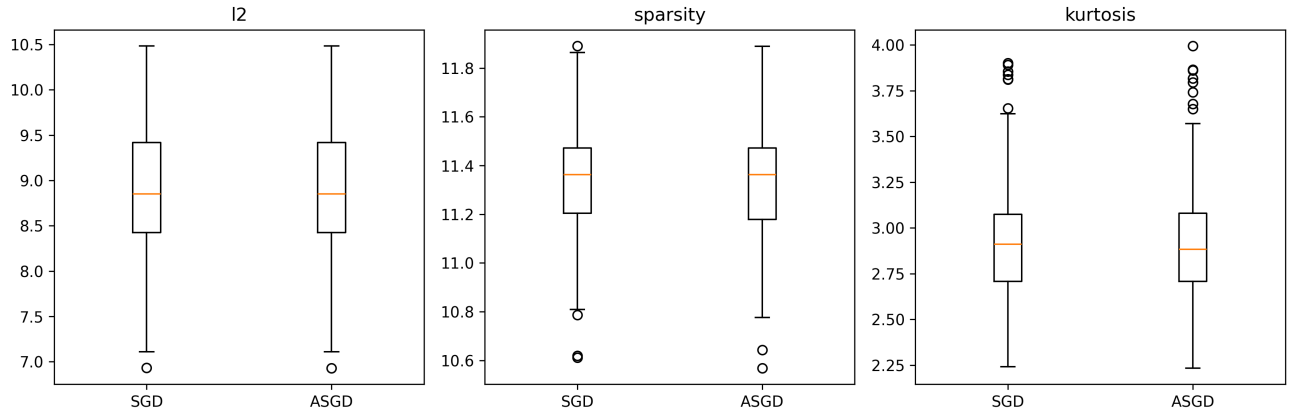


Figure 15. Box plots of per-run weight statistics for SGD vs. SA-ASGD under 100% over-parametrization. Hollow circles denote individual runs falling beyond 1.5 × IQR from the 25th or 75th percentile (statistical outliers).

D. Convergence Assumptions

D.1. Lipschitz Gradients

$$\mathbb{E}[\|\nabla L(x) - \nabla L(y)\|] \leq \zeta \mathbb{E}[\|x - y\|] \tag{1}$$

D.2. Bounded Gradient Moment

$$\mathbb{E}[\|\nabla L(x)\|^2] \leq M^2 \tag{2}$$

D.3. μ-strong convexity

$$L(x) - L(y) \geq \langle \nabla L(y), x - y \rangle + \frac{\mu}{2}\|x - y\|_2^2 \tag{3}$$

D.4. Bounded Staleness

$$0 \leq \tau \leq 2n \tag{4}$$

Condition regarding the maximum staleness τ of any gradient to be at most $2n$, where n is the number of workers. It is shown empirically in (Zhang et al., 2016) that staleness is close to n and biggerthan $2n$ with a negligible probability (less than 0.00001).

D.5. Learning Rate Upper Bound for SA-ASGD

$$\alpha_0 \leq \frac{c\, C_2\, p_t}{C_3 \sum_{j=t-2n}^{t-1} \frac{1}{p_j}} \tag{5}$$

The property ensures that the learning rate α_0 used is small enough relative to the staleness. In the formula c is the number of gradients used for updated (so the total aggregation size), C_2, C_3 are problem-dependent constants and p_t is the effective staleness at update step t.

D.6. Stability Condition for SA-ASGD accumulated delayed update

$$\frac{C_3\, \alpha_0}{c\, p_t} + \frac{C_4\, n\, \alpha_0^2}{c^2\, p_t} \sum_{\kappa=1}^{2n} \frac{1}{p_{t+\kappa}} \leq 1 \tag{6}$$

Guarantees that the effect of stale and accumulated updates does not destabilize training. In the formula n is a value chosen to bound the overall staleness window (staleness control parameter), C_3 and C_4 bounds the effect of each gradient on the loss and the effect of effect of gradient noise and staleness coupling respectively, p_t is the effective staleness at update step t, c is the number of gradients aggregated per server update and α_0 is the base learning rate (that needs to be tuned).