Exploring Learning Rate Scaling Rules for Distributed ML Training on Transient Resources

Joel André *
joel.andre@inf.ethz.ch
ETH Zurich

Foteini Strati *
foteini.strati@inf.ethz.ch
ETH Zurich

Ana Klimovic
aklimovic@ethz.ch
ETH Zurich

ABSTRACT

Training Machine Learning (ML) models to convergence is a long-running and expensive procedure, as it requires large clusters of high-end accelerators such as GPUs and TPUs. Many ML frameworks have proposed *elastic* distributed training, which enables using transient resources such as spot VMs in the cloud, reducing the overall cost. However, the availability of transient resources varies over time, creating an inherently dynamic environment that requires special handling of training hyperparameters. Techniques such as gradient accumulation enable using the same hyperparameters upon resource preemptions, however sequentially accumulating gradients stalls synchronous distributed training. On the other hand, scaling the batch size according to the available resources requires tuning of other hyperparameters, such as the learning rate. In this work, we study how learning rate scaling rules perform under dynamic environments when the batch size changes frequently and drastically, as we observed in real cloud clusters. We build a PyTorch-based system to evaluate Stochastic Gradient Descent on Image Recognition and Object Detection tasks under various learning rate scaling rules and resource availability traces. We observe minor or no degradation in model convergence when choosing the correct learning rate scaling rule. Identifying the appropriate scaling rule for a given model is non-trivial. Automating this decision remains an open question.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Elastic Deep Learning, Distributed Training, Spot Instances

ACM Reference Format:
Joel André, Foteini Strati, and Ana Klimovic. 2022. Exploring Learning Rate Scaling Rules for Distributed ML Training on Transient Resources. In *DistributedML (DistributedML '22), December 9, 2022, Roma, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3565010.3569067

*Equal Contribution.

DistributedML '22, December 9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9922-7/22/12…$15.00
https://doi.org/10.1145/3565010.3569067

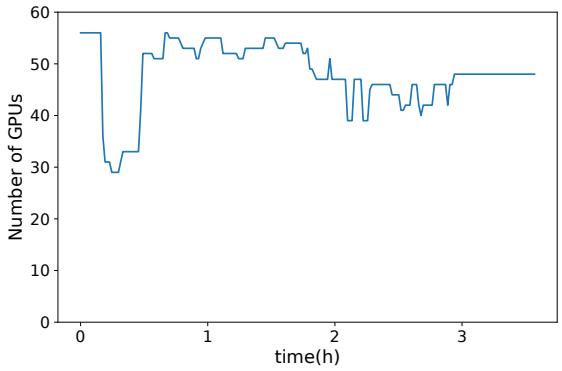
1 INTRODUCTION

Recent advances in Machine Learning (ML) research have led to the creation of complex models, trained with massive datasets [1, 2]. In order to excel at their tasks, state-of-the-art ML models are trained for long periods of time in a distributed way using large clusters with high-end accelerators such as GPUs [3] and TPUs [4]. This leads to very high costs [5], since training can take hours or even days to complete, even in enormous clusters. Furthermore, most distributed ML frameworks [6, 7], consider a static, *all-or-nothing* allocation of resources for distributed training, meaning that a training job only gets scheduled when all of its requested resources are available. This leads to long queuing of training jobs in multi-tenant clusters, especially when they request a vast amount of resources [8, 9].

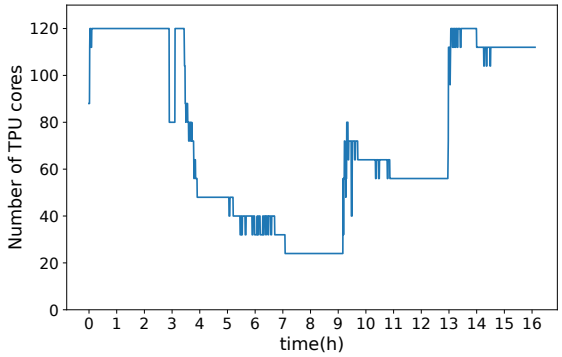
In response, ML training frameworks [10–15] are becoming *elastic*, meaning that they are able to train over a dynamic set of resources. Jobs can get scheduled before all requested resources are available, reducing the queuing time of jobs in multi-tenant clusters, and increasing the overall cluster resource utilization [16]. Furthermore, elastic ML frameworks enable reducing training costs by leveraging *spot* Virtual Machines (VMs) offered by cloud providers [17–19] at a 60%-90% discount. In this setting, the resources of a training job could be preempted and reclaimed multiple times during training, according to cluster resource contention. For example, Figure 1 shows how frequently and abruptly the number of available preemptible hardware accelerators can change on Google Cloud.

One of the most challenging aspects of elastic training is dealing with training hyperparameters as the number of underlying resources changes. In particular, when the underlying resources available for training decrease due to preemptions, naively continuing to train without accounting for preempted nodes reduces the effective batch size of the training job. Varying the batch size during training affects model accuracy and requires careful tuning of other hyperparameters. Some elastic frameworks [12] decouple the global batch size from the available resources and employ techniques such as gradient accumulation to maintain the same hyperparameters, which can help guarantee convergence consistency of elastic training jobs. However, this can introduce stragglers [20] in the case of uneven partitioning of the same amount of work in fewer resources, which reduces synchronous training throughput.

Other frameworks [10, 11, 14] employ a simpler approach: scaling the global batch size according to the number of available resources. This requires users to decide on how to scale hyperparameters, such as the learning rate, in response to batch size changes. However, since users cannot usually predict how the hyperparameter changes will affect the training dynamics and we lack sufficient empirical



(a) A100 GPUs (a2-highgpu) in europe-west4-a



(b) TPU cores (TPUv2-8) in us-central1-f

Figure 1: Availability of preemptible accelerators in Google Cloud Platform.

studies and theoretical convergence guarantees, this approach is rarely used in practice [9, 12].

In this work, we take a first step in studying the effects of dynamically scaling the batch size and learning rate based on the number of available resources, and empirically understanding whether this is a viable solution for elastic training. We focus on the learning rate, due to its heavy influence on training convergence and model generalization ability [21–23]. A wide variety of learning rate scaling rules have been proposed, to enable training with large batch sizes. However, these rules consider a static allocation of resources (and thus a static global batch size) throughout training, and they have not been evaluated in environments where the global batch size changes frequently and drastically, such as cloud-based training using spot VMs. In this work, we explore the applicability of these scaling rules in such volatile environments. We study widely used learning rate scaling rules: linear, square-root, and AdaScale. We use Image Recognition and Object Detection tasks for our study and collect traces of spot instances on Google Cloud as representative environments with dynamic resource availability.

Our preliminary results show that when the correct learning rate scaling rule is employed, elastic training can lead to minimal, or no degradation in model convergence. However, the correct rule depends on the type of model and it is difficult to know which scaling rule to apply. Our work motivates further exploration to understand the influence of elasticity across diverse ML jobs and automate learning rate scaling rule selection for users.

In summary, our work has two key contributions.

(1) We implement a PyTorch-based framework that enables the evaluation of learning rate scaling rules in environments with dynamic resource availability. Users can simply plug in various learning rate scaling rules and arbitrary resource availability traces, and evaluate how well these rules perform under the resource variations modeled by the given traces.
(2) We empirically evaluate how SGD performs in environments with frequent resource changes, using well-known and widely used learning rate scaling rules. We base our evaluation on real transient VM traces collected from Google Cloud.

2 BACKGROUND

2.1 Transient VMs in the cloud

Transient, or low-priority, preemptible resources are found in the cloud [17–19] as well as in private datacenters [8, 24]. Our study focuses on transient cloud VMs, however, our analysis is valid for any environment with preemptible resources.

Transient or *spot* VMs are offered by most cloud providers today and can lead to large cost savings (up to 60-90%). However, they can be preempted at any point in time, depending on the resource demand, leading to sudden failures. Their availability is heavily related to the cluster-wide resource contention and each provider's policy. Figures 1a and 1b characterize the availability of spot GPU and TPU VMs in Google Cloud, respectively. The traces were collected in April 2022. For GPU trace collection we used `a2-highgpu` VMs with A100 GPUs attached located in zone `europe-west4-a` and we try to maintain a constant amount of 56 GPUs using GKE node pools [25]. For TPU trace collection we used `TPUv2-8` nodes in zone `us-central1-f` with 8 TPU cores each, and try to maintain 120 TPU cores, meaning that whenever a machine is preempted, we try to allocate a new one.

From Figure 1a, and 1b, we see that the available number of preemptible accelerators varies over time. Node preemption and reclamation events are unpredictable and can happen in bulks (e.g. see Figure 1b at time = 4h, and time = 9h), leading to a dynamic environment. Prior work [13] has also observed high volatility of transient resources in the cloud.

2.2 Learning rate scaling rules in SGD

We refer to the number of samples given as input in each accelerator during the forward and backward pass as the *local* batch size. The *global* batch size in synchronous training is defined as the number of samples taken into account for the optimizer update (e.g. Stochastic Gradient Descent (SGD)) and is typically the sum of all accelerators' local batch sizes [20]. The impact of the optimization step is heavily determined by the learning rate.

In SGD, the global batch size is tightly coupled with the learning rate hyperparameter, as has been shown in recent studies [21]. Scaling the batch size usually requires scaling the learning rate in order not to degrade model quality. Multiple learning rate scaling rules have been proposed to achieve good convergence with large batch sizes. Here, we present three widely used rules which we

evaluate in this work, due to their wide applicability and adoption by the ML research community [26–28], and ease of use [29].

- **Linear Rule** According to the linear scaling rule, when the batch size is scaled by a factor k, the learning rate should also scale by k. Goyal et al. [26] give an intuitive interpretation of this rule stating that if the gradients of k steps with batch size n are identical to the gradients of 1 step with batch size $k \cdot n$, they would lead to equivalent weights, if the learning rate is also scaled by k. Even though this is a strong assumption, this rule has been widely used to scale distributed SGD in various ML tasks [27, 30].
- **Square Root Rule** The square root scaling rule states that when the batch size is scaled by a factor k, the learning rate should scale by $\sqrt{k}$. Krizhevsky [31] proposed this rule in order to maintain the scale of weight update variance. Variations of this rule have been extensively used especially with the Adam optimizer [32].
- **AdaScale Rule** The Adascale rule, proposed by Johnson, Agrawal et al [33], adapts the learning rate scale depending on the variance of the gradient. If the batch size is scaled by a factor k, when the gradient's variance is large, Adascale approximates the linear scaling rule (learning rate scale close to k), while when the variance is small, the scale of the learning rate is also smaller, or even 1. Thus, Adascale translates into a learning rate warm-up scheme, and has enabled training with very large batch sizes in a wide variety of ML tasks [33].

These rules can efficiently scale to larger batch sizes, reducing the total number of steps needed to reach convergence, and thus reducing the total training time. However, they have been originally proposed considering a constant batch size throughout training. As shown in section 2.1, training with transient resources can lead to frequent and drastic variation in the global batch size, raising the question of whether these well-established rules hold in these scenarios. To the best of our knowledge, the effect of these rules has not been adequately studied in environments where the batch size can change so frequently and abruptly.

3 RELATED WORK

Batch size and learning rate at scale: Shallue, Lee et al. [21] conduct a detailed empirical study of the effects of large batch size on training convergence, using models from various ML tasks. They investigate the optimal learning rate for each batch size scale and observe huge variations among different workloads, as well as discrepancies between the actual optimal learning rates, and the learning rates proposed by scaling rules. Zhang et al. [22] use a noisy quadratic model to predict the effects of a larger batch size as well as the optimal learning rate when the batch size is scaled, while techniques such as LARS [34] and LAMB [23] have managed to effectively scale the batch size even further in image processing and Natural Language Processing tasks. These works study the effect of learning rates with a static amount of resources throughout training. Lin et al. [30] study the impact of dynamic batch size in SGD with momentum and propose a *momentum compensation* technique that gradually changes the learning rate when the batch size changes, but they only consider the linear learning rate rule. Since training is becoming elastic, and the batch size can change very frequently, we believe that it is important to extend these studies to extremely dynamic environments as well.

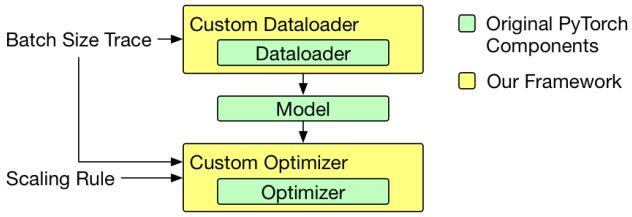


Figure 2: Evaluation Framework

Frameworks for elastic training: TorchElastic [10] and Horovod Elastic [14] are systems proposed by well-known ML frameworks to support elastic training. Pollux [11] leverages elastic training to optimize resource efficiency and average job completion time in a cluster, by scaling a training job up or down based on its throughput and convergence rate. In all these frameworks, the user needs to decide how the learning rate will change in response to batch size changes, usually by defining plugins in the training procedure. EasyScale [15] proposes a system for deterministic elastic training that finds and mitigates the sources of randomness in training. However, this may negatively impact throughput, as vendor-specific kernel optimizations that speed up training are disabled. Recently, Varuna [12] and Bamboo [13] enable elasticity for large models that are trained using pipeline parallelism. Varuna [12] uses gradient accumulation to preserve the global batch size but may change the local batch size, leading either to hardware inefficiencies [35], or to batch normalization issues, requiring techniques such as Group Normalization [36].

4 METHODOLOGY

This section describes how we evaluate different learning rate scaling rules with different models. We evaluate the scaling rules by training ResNet50 [37] and VGG16 [38] on the ImageNet-2012 [39] dataset and Single Shot MultiBox Detector (SSD) [40] on the MS-COCO [41] object detection dataset. We implement a framework that emulates training with synchronous SGD in an elastic environment, where the batch size varies according to hardware resource availability. The framework makes it easy to plug in new learning rate scaling rules, thus enabling future research in this direction, and is available online [1].

4.1 Framework Design

Our framework lets users specify a learning rate scaling rule, a local (per-accelerator) batch size, and a batch size trace. The trace can represent an arbitrary elastic environment. The framework then trains using a dynamic batch size as specified by the trace, and adapts the learning rate upon batch size changes as defined by the given scaling rule. The framework itself runs on a cluster of normal/non-preemptible resources.

The framework is designed to run on top of PyTorch. As we use TPU accelerators for our experiments, our framework is designed to

[1] https://github.com/eth-easl/elastic-learning-rate-evaluation

run with PyTorch-XLA [42]. Any Pytorch XLA training script can be modified to use our framework by adding our custom sampler class and wrapping the PyTorch optimizer with one of our custom optimizers.

Since we only want to evaluate the effect of learning rate scaling rules, we assume that no data samples are lost when resources change. This models the case where the dataset can be redistributed to the active workers after a resource change. We further design our system to preserve the exactly-once semantics, for the dataset at each training epoch, regardless of the number of resources. Finally, our framework requires the trace to specify the batch size for each step. In contrast, the resource availability measurements shown in Figure 1 display the number of resources over time. Hence raw measurement traces must be translated. System characteristics such as throughput, scaling efficiency, and per-accelerator batch size must be taken into account during translation. By customizing these parameters our framework can emulate arbitrary cluster configurations. Note that we assume that the trace is generated such that the global batch size is always a multiple of the local batch size that is used per accelerator. Therefore, it is guaranteed that the global batch size is always greater or equal to the local batch size.

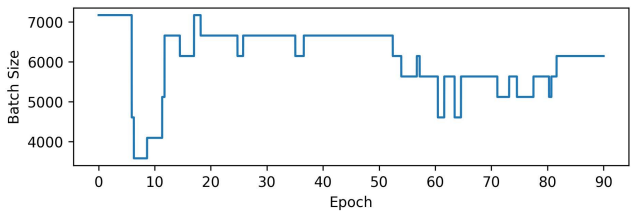
In our experiments we assume that the time per step Δt is constant independently of the batch size (perfect scaling). In this case the batch size trace can be computed from the trace of available resources over time as follows: The resource availability trace defines a function g : time $\rightarrow$ #accelerators. The batch size at step i is $g(i \cdot \Delta t + T) \cdot B$, where T is an optional starting offset and B is the local per-accelerator batch size.

Implementation Details. We scale the global batch size according to the given trace by dynamically adjusting the number of accelerators involved in a step and accumulating gradients if needed. The local batch size stays constant. In other words, the number of accumulated local batches at each accelerator varies dynamically during training. Our framework takes care of distributing the right batches to accelerators so that each sample is seen once per epoch.

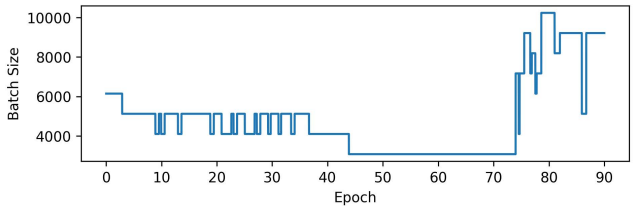
4.2 Batch Size Traces

We evaluate the scaling rules with traces based on data gathered from real GPU and TPU spot instances in Google Cloud (Figure 1a and 1b). We choose starting offsets so that the resulting traces are both representative and interesting.

ResNet50 and VGG16. For both models we assume that each accelerator (GPU or TPU core) can efficiently handle a local batch size of 128. That is, a batch size of 1024 in a trace indicates that eight accelerators were available at this point in time. For simplicity, we assume that each step takes 0.6s independently of the batch size. This is a value that we got from running the two models under various GPU cluster configurations. With this configuration, a training run of 90 epochs on ImageNet-2012 with the traces corresponds to 200 to 275 minutes of cloud availability measurements. Fig. 3a corresponds to the batch size trace based on available A100 GPUs (Figure 1a), and Figure 3b shows the trace sampled from the the



(a) GPU Trace



(b) TPU Trace

Figure 3: Batch size trace for models trained on ImageNet

TPU v2-8 availability data (Figure 1b)

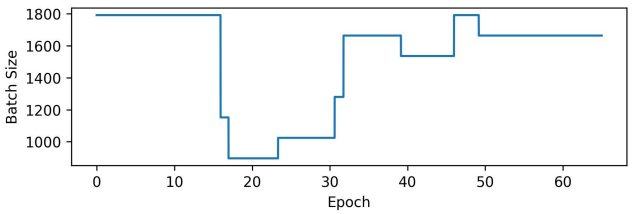
SSD. We assume that each accelerator handles a batch size of 32. Each step takes 0.3s independently of the batch size. The resulting batch size traces are shown in Figure 4.

Reproducibility. The traces that we used for our experiments can be considered *snapshots* of the GCP resource availability. In general, there are multiple sources of randomness in ML training, such as multi-worker data loading, hardware-specific implementations, and the non-associative nature of floating point operations. Elasticity adds another source of randomness in training, since the experienced resource availability varies over time. Related work [15] attempts to enforce reproducibility in elastic training, at the cost of increased runtime.

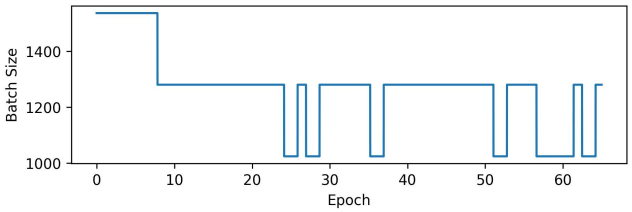
4.3 Model Training Configurations

ResNet50. We train ResNet50 according the training procedure described in [26]. The model is trained for 90 epochs. The learning rate is linearly warmed up during the first five epochs. At epochs 30, 60, and 80, the learning rate is scaled by 1/10. The reference learning rate used for the learning rate scaling rules is 0.1 for batch size 256. The local batch size is always 32. As a baseline for the linear scaling rule, we reproduce the result of [26] by training with batch size 8192 and a linearly scaled learning rate ($8192/256 \cdot 0.1 = 3.2$). Goyal et al.[26] report a Top-1 accuracy of 73.27% for static batch size 8192. For the root scaling rule, batch size 8192 implies a learning rate of $\sqrt{8192/256} \cdot 0.1 = 0.57$.

VGG16. We opt to evaluate a modified version of VGG16 that includes batch-normalization as defined in [43]. We train VGG16 with the same training procedure as ResNet50, which also follows the official PyTorch recipe [44]. We again establish a baseline using batch size 8192. In our experiments, the reference learning rate of 0.1 for batch size 256 leads to exploding gradients when linearly scaled to



(a) GPU Trace



(b) TPU Trace

Figure 4: Batch size trace for models trained on MS-COCO

batch size 8192. Hence, for the linear scaling rule, we change the reference learning rate so that the learning rate obtained with the linear scaling rule is equivalent to the learning rate obtained with the root scaling rule at batch size 8192. That is, we use reference learning rate 0.0177 instead of 0.1 (at batch size 256) for the linear scaling rule. Note that for batch sizes below 8192, the linear scaling rule will thereby dictate a lower learning rate than the root scaling rule. The reference learning rate for AdaScale is changed identically.

SSD. Single Shot MultiBox Detector[40] is an object detection network. We use the PyTorch implementation given in [45]. As a backbone the network internally uses ResNet50 with weights pre-trained on ImageNet. We train the model using the MS-COCO[41] dataset. We use the 118k samples from *train2017* to train and validate our model using the *val2017* dataset. Images are resized to 300x300px. The model is trained for 65 epochs as described in [45]. The reference learning rate for scaling is 2.7e-3 for batch size 32. The learning rate is decayed by 1/10 at epochs at 43 and 54 and linear warmup is used for the first 1000 steps. As evaluation metric we use *mAP[0.5:0.95]* as used in [40]. mAP[0.5:0.95] is defined as the average mAP over different IoU thresholds ranging from 0.5 to 0.95. From here on we refer to this metric simply as mAP. [45] report a final mAP of 0.25-0.26 when training SSD on MS-COCO for different batch sizes ranging from 128 to 2048.

5 EVALUATION

In this section we present the results of our empirical evaluation of how the presented learning rate scaling rules perform when the batch size varies during training. We evaluate all three scaling rules, on all batch size traces with all models, except for SSD. We omit results with AdaScale on SSD since training did not properly converge in our experiments. All experiments are run with full precision on TPUv3-8 virtual machines using all eight cores. Each experiment is run three times, and the reported numbers are the mean and standard deviation of the runs. For ResNet50 and VGG16, each run's reported accuracy is the mean of the last three epochs' validation accuracy. To get a precise baseline for ResNet50 and VGG16 we evaluate the rules using a static 8192 batch size. For SSD on the MS-COCO dataset we consider the results in [45], where 0.25-0.26 mAP is reached for different batch sizes, to be our baseline. The results are presented in tables 1, 2, 3.

Trace	Linear Rule	Root Rule	AdaScale
Static	76.02 ± 0.11	73.70 ± 0.14	**76.31 ± 0.18**
GPU	76.15 ± 0.12	74.38 ± 0.07	**76.39 ± 0.23**
TPU	76.20 ± 0.10	74.84 ± 0.19	**76.31 ± 0.03**

Table 1: Final Top-1 (validation) accuracy of ResNet50 for all combinations of traces and scaling rules. Standard deviation displayed based on three runs.

Trace	Linear Rule	Root Rule	AdaScale
Static	-	**72.62 ± 0.14**	72.37 ± 0.10
GPU	72.71 ± 0.11	**73.05 ± 0.12**	72.37 ± 0.03
TPU	72.63 ± 0.09	**73.21 ± 0.14**	72.58 ± 0.10

Table 2: Final Top-1 (validation) accuracy of VGG16 for combinations of traces and scaling rules. Standard deviation displayed based on three runs.

Trace	Linear Rule	Root Rule
Static	0.25-0.26	
GPU	**0.247 ± 0.002**	0.200 ± 0.001
TPU	**0.250 ± 0.001**	0.204 ± 0.001

Table 3: Final Top-1 (validation) mAP[0.5:0.95] of SSD300 for combinations of traces and scaling rules. Standard deviation displayed based on three runs.

Impact on final accuracy ResNet50. Our results show that for ResNet50, model quality does not degrade with the linear scaling rule on both traces. The root scaling rules performs worse on both traces, but final accuracy is inferior with the baseline too. The learning rate computed by the root scaling rule is most likely too small for this setting. The AdaScale optimizer outperforms both other rules on all traces.

Impact on final accuracy VGG16. In contrast to our observations with ResNet50, the linear learning rate scaling rule does not perform well with the VGG architecture. If we use the same reference learning rate as for the root scaling rule to scale the batch size, training loss diverges. Yet, even if we reduce the learning rate as described in 4.3, the linear learning rate performs worse on all traces. The root learning rate, on the other hand, has a convergent baseline, and model quality does not degrade on any of the traces.

We hypothesize that smaller batch sizes help training convergence more than batch size changes harm it. AdaScale performs worse than the root scaling rule. Like the linear scaling rule, AdaScale training did not properly converge with the standard learning rate. With the adjusted learning rate, the final accuracy is still inferior to what we observe with the root scaling rule.

Impact on final mAP SSD. We observe that for SSD training the linear scaling rule results in minimal model quality degradation. The root scaling rule performs inferiorly.

Impact on training curves. In Figure 5 we further display the training curves for all combinations of rules and traces on ResNet50. Note that the batch size changes are hardly visible on the training curve plots.

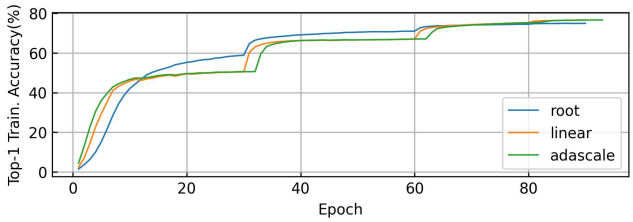
Takeaways. Overall, we observe that the best learning rate scaling rule is workload-dependent. The fact that linear scaling works better with ResNet50 and root scaling is better with VGG16 matches the results in [21], where Shallue et al. empirically investigate the optimal learning rate for different batch sizes and various models. They find that for ResNet50 the optimal learning rate closely follows the linear scaling rule and for VGG16 the optimal learning rate is closer to the root scaling rule. Hence if practitioners care about not affecting training dynamics, they should profile their model to find the appropriate scaling rule. Our preliminary results show that, for some jobs, where accuracy drops of 1-3% are tolerable, compensating for changing batch size with an appropriate scaling rule is sufficient.

Cost. Using transient resources in the cloud can bring significant cost reductions, which might compensate for small accuracy drops. For example, according to our simulations, training ResNet50 on ImageNet to convergence in a cluster of 64 V100 on-demand GPUs costs around 344 USD. On the other hand, training the same model over spot V100 GPUs (around 70% cheaper than on-demand [19]), can reduce the training cost to 112 USD if the GPU availability varies according to the trace 1a. However, as the results from the three different models show, users might need to profile the convergence of their models under the various learning rate rules if they wish to use spot VMs and let batch size scale accordingly, which might add up to the training cost.

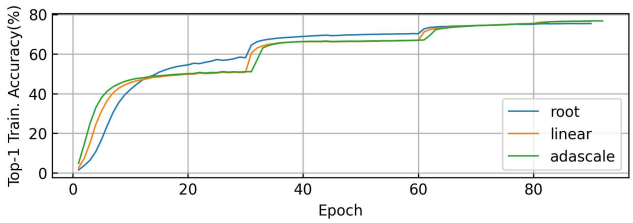
6 LIMITATIONS AND FUTURE WORK

Although in this work we focused on Image Classification and Object Detection tasks, extending our analysis with more models from diverse domains would help build a more in-depth understanding of the relationship between learning rate and batch sizes in volatile environments. Furthermore, layerwise adaptive learning rates scaling rules such as LARS [34] and LAMB [23] could be evaluated, as they have enabled training with very large batch sizes and are widely used [46]. Our system enables further experimentation, since users can simply plug in new learning rate rules and arbitrary availability traces.

Our study also assumes no sample dropping, meaning that all samples in a dataset are always available, regardless of node failures. This is true for GB-sized datasets such as ImageNet [39] and WikiCorpus [47], but may not hold for TB-sized datasets [2], where samples might be physically partitioned across nodes. In that case, some samples might become unreachable after a node failure, leading to sample dropping [13]. In cases where dropping training examples does not harm model quality, it could lead to lower training time [48] compared to dataset repartitioning. For future work, we plan to evaluate how sample dropping affects the training in elastic environments when matched with dynamic batch size.



(a) GPU Trace



(b) TPU Trace

Figure 5: Training set accuracy convergence of ResNet50 using different scaling rules for each trace (Single Run)

7 CONCLUSION

In this work, we explored the effect of various learning rate scaling rules when training using transient resources, where the batch size is scaled according to resource availability. We implemented a PyTorch-based framework that enables evaluating learning rate scaling rules over arbitrary traces of dynamic batch sizes and used it to evaluate widely used rules in Object Detection and Image Recognition tasks. Our experiments show that when the correct learning rate scaling rule is employed, training is hardly (or not at all) affected by batch size variability. However, there is no one-size-fits-all rule, and users need to profile their model for selecting the most appropriate one. Alternatively, techniques such as gradient accumulation should be used to avoid changing hyperparameters, but they can introduce extra complexity or inefficiency. In future work, we plan to investigate a wider range of models and learning rate scaling rules to better understand the effects of elasticity on model accuracy for diverse ML tasks.

ACKNOWLEDGEMENT

We thank our anonymous reviewers for their valuable feedback. We also thank Google TPU Research Cloud for giving us free access to cloud TPUs.

REFERENCES

[1] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, mar 2020. [Online]. Available: https://doi.org/10.1007%2Fs11263-020-01316-z

[2] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Karna, D. Moise, S. J. Pennycook, K. Maschoff, J. Sewall, N. Kumar, S. Ho, M. Ringenburg, Prabhat, and V. Lee, "Cosmoflow: Using deep learning to learn the universe at scale," 2018. [Online]. Available: https://arxiv.org/abs/1808.04728

[3] "Meta works with nvidia to build massive ai research supercomputer," https://blogs.nvidia.com/blog/2022/01/24/meta-ai-supercomputer-dgx/, accessed: 2022-09-10.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, jun 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[5] O. Sharir, B. Peleg, and Y. Shoham, "The cost of training nlp models: A concise overview," 2020. [Online]. Available: https://arxiv.org/abs/2004.08900

[6] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018. [Online]. Available: https://arxiv.org/abs/1802.05799

[7] A. Paszke, 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, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[8] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 947–960. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/jeon

[9] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 945–960. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/weng

[10] "Torch distributed elastic," https://pytorch.org/docs/stable/distributed.elastic.html, accessed: 2022-09-10.

[11] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/qiao

[12] S. Athlur, N. Saran, M. Sivathanu, R. Ramjee, and N. Kwatra, "Varuna: Scalable, low-cost training of massive deep learning models," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 472–487. [Online]. Available: https://doi.org/10.1145/3492321.3519584

[13] J. Thorpe, P. Zhao, J. Eyolfson, Y. Qiao, Z. Jia, M. Zhang, R. Netravali, and G. H. Xu, "Bamboo: Making preemptible instances resilient for affordable training of large dnns," 2022. [Online]. Available: https://arxiv.org/abs/2204.12013

[14] "Horovod elastic," https://horovod.readthedocs.io/en/stable/elastic_include.html, accessed: 2022-09-10.

[15] M. Li, W. Xiao, B. Sun, H. Zhao, H. Yang, S. Ren, Z. Luan, X. Jia, Y. Liu, Y. Li, D. Qian, and W. Lin, "Easyscale: Accuracy-consistent elastic training for deep learning," 2022. [Online]. Available: https://arxiv.org/abs/2208.14228

[16] C. Hwang, T. Kim, S. Kim, J. Shin, and K. Park, "Elastic resource sharing for distributed deep learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 721–739. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/hwang

[17] "Amazon ec2 spot," https://aws.amazon.com/ec2/spot/, accessed: 2022-09-10.

[18] "Azure spot virtual machines," https://azure.microsoft.com/en-us/services/virtual-machines/spot/#overview, accessed: 2022-09-10.

[19] "Google cloud spot vms," https://cloud.google.com/compute/docs/instances/spot, accessed: 2022-09-10.

[20] A. Or, H. Zhang, and M. N. Freedman, "Virtualflow: Decoupling deep learning models from the underlying hardware," in *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu, Eds., vol. 4, 2022, pp. 126–140. [Online]. Available: https://proceedings.mlsys.org/paper/2022/file/2723d092b63885e0d7c260cc007e8b9d-Paper.pdf

[21] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," 2018. [Online]. Available: https://arxiv.org/abs/1811.03600

[22] G. Zhang, L. Li, Z. Nado, J. Martens, S. Sachdeva, G. E. Dahl, C. J. Shallue, and R. Grosse, *Which Algorithmic Choices Matter at Which Batch Sizes? Insights from a Noisy Quadratic Model*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[23] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," 2019. [Online]. Available: https://arxiv.org/abs/1904.00962

[24] P. Sharma, A. Ali-Eldin, and P. Shenoy, "Resource deflation: A new approach for transient resource reclamation," ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303945

[25] "Gke node pools," https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools, accessed: 2022-09-10.

[26] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," 2017. [Online]. Available: https://arxiv.org/abs/1706.02677

[27] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun, "Megdet: A large mini-batch object detector," 2017. [Online]. Available: https://arxiv.org/abs/1711.07240

[28] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: Closing the generalization gap in large batch training of neural networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1729–1739.

[29] "Fairscale," https://github.com/facebookresearch/fairscale, accessed: 2022-09-10.

[30] H. Lin, H. Zhang, Y. Ma, T. He, Z. Zhang, S. Zha, and M. Li, "Dynamic mini-batch sgd for elastic distributed training: Learning in the limbo of resources," 2019. [Online]. Available: https://arxiv.org/abs/1904.12043

[31] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1404.5997

[32] Y. You, J. Hseu, C. Ying, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large-batch training for lstm and beyond," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356137

[33] T. B. Johnson, P. Agrawal, H. Gu, and C. Guestrin, "Adascale sgd: A user-friendly algorithm for distributed training," 2020. [Online]. Available: https://arxiv.org/abs/2007.05105

[34] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," 2019. [Online]. Available: https://arxiv.org/abs/1904.00962

[35] "Troubleshooting tensorflow - tpu," https://cloud.google.com/tpu/docs/troubleshooting/trouble-tf#batch-too-small, accessed: 2022-09-10.

[36] Y. Wu and K. He, "Group normalization," 2018. [Online]. Available: https://arxiv.org/abs/1803.08494

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[40] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, pp. 21–37.

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[42] "Pytorch xla package," https://github.com/pytorch/xla/, accessed: 2022-09-10.

[43] "Pytorch vgg-bn implementation," https://pytorch.org/vision/stable/models/generated/torchvision.models.vgg16_bn.html, accessed: 2022-09-10.

[44] "Pytorch: Vgg pretraining recipe," https://github.com/pytorch/vision/tree/main/references/classification, accessed: 2022-09-10.

[45] "Nvidia deep learning examples: Ssd," https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Detection/SSD, accessed: 2022-09-20.

[46] T. Wang, Y. Zhu, C. Zhao, W. Zeng, Y. Wang, J. Wang, and M. Tang, "Large batch optimization for object detection: Training coco in 12 minutes," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 481–496.

[Online]. Available: https://doi.org/10.1007/978-3-030-58589-1_29
[47] "Wikipedia datasets," https://en.wikipedia.org/wiki/Wikipedia:Database_download, accessed: 2022-09-22.
[48] T. Wang, J. Huan, and B. Li, "Data dropout: Optimizing training data for convolutional neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1809.00193