Dynamic Mini-batch SGD for Elastic Distributed Training: Learning in the Limbo of Resources

Haibin Lin, Hang Zhang, Yifei Ma, Tong He, Zhi Zhang, Sheng Zha, Mu Li
Amazon Web Services
Palo Alto, CA
{haibilin, hzaws, yifeim, htong, zhiz, zhasheng, mli}@amazon.com

Abstract

With an increasing demand for training powers for deep learning algorithms and the rapid growth of computation resources in data centers, it is desirable to dynamically schedule different distributed deep learning tasks to maximize resource utilization and reduce cost. In this process, different tasks may receive varying numbers of machines at different time, a setting we call elastic distributed training. Despite the recent successes in large mini-batch distributed training, these methods are rarely tested in elastic distributed training environments and suffer degraded performance in our experiments, when we adjust the learning rate linearly immediately with respect to the batch size. One difficulty we observe is that the noise in the stochastic momentum estimation is accumulated over time and will have delayed effects when the batch size changes. We therefore propose to smoothly adjust the learning rate over time to alleviate the influence of the noisy momentum estimation. Our experiments on image classification, object detection and semantic segmentation have demonstrated that our proposed Dynamic SGD method achieves stabilized performance when varying the number of GPUs from 8 to 128. We also provide theoretical understanding on the optimality of linear learning rate scheduling and the effects of stochastic momentum.

1. Introduction

Deep learning has yet become the de facto standard algorithm in computer vision. Deeper and larger models keep boosting the state-of-the-art performance in image classification [13, 41], object detection [37, 26] and semantic segmentation [27, 3]. In addition, computation-intensive tasks such as video classification [45, 8], segmentation [24, 44] and visual question answering [43, 22] bring more interests to the community as the more computation resources become available. Despite the rapid growth of the computation powers in the data centers of research labs and leading IT companies, the demand of training deep learning models can be barely satisfied. To train a job using multiple machines, we may need to wait for a long time for the job to start due to limited resources. To finish high-priority training jobs in time, we may reserve resources in advance or stop other low-priority jobs.

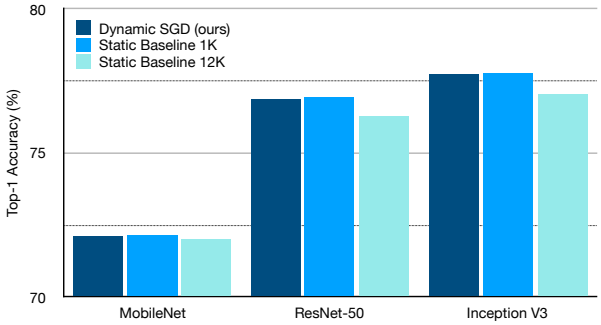


Figure 1: Top-1 accuracy on ImageNet dataset *vs.* different training methods. Static baseline refers to training with mini-batch size and number of machines fixed. In elastic distributed training environments, the mini-batch size is dynamically updated. The proposed Dynamic SGD method enables stable training in such dynamic environments while benefits from the speedup with elastic resources. Our elastic training method achieves comparable model accuracy with small mini-batch training (1K), and always outperforms the large mini-batch one (12K) under the same setting.

Allowing resources for a training job to change dynamically can greatly reduce the waiting time and improve resource utilization. We could start a job earlier when a portion of the requested resources is ready. We could also preempt resources from running low-priority training jobs without stopping them for high-priority tasks. In addition, cloud providers encourage users to use preemptible resources with significant lower prices, such as spot instance in AWS [39], low-priority virtual machines in Azure [31] and preemptible virtual machines on Google Cloud [10]. We refer to the above dynamic machine environment where the machines

could be added or removed from a task as an elastic distributed training environment.

Mini-batch stochastic gradient descent (SGD) with momentum (e.g. heavy-ball momentum, Nesterov momentum, Adam [33, 20]) is a widely used optimization method to train deep learning models in computer vision. Previous work focuses on asynchronous SGD (asyncSGD) for environments where machines are loosely coupled [5]. AsyncSGD, however, converges radically differently from synchronous mini-batch SGD [49, 36, 52]. It is also difficult for asyncSGD to match the model accuracy trained with synchronized mini-batch SGD with similar computation budget [5, 2].

In this work, we focus on extending synchronized mini-batch SGD with momentum into the dynamic training environment. We aim to minimize the convergence difference when the training environment is constantly changing. There are two straightforward approaches to extend synchronized SGD with momentum. One approach is to fix the mini-batch size but reassign mini-batches across machines when the number of machines changes. With the mini-batch size fixed, the number of machines has little impact on the optimization method, but reduces the computational efficiency when adding many machines, since the assigned per machine mini-batch size decreases linearly. The other approach fixes the mini-batch size per machine, and updates the total mini-batch size linearly with the number of machines, while linearly changing the learning rate at the same time [11, 7, 40]. Empirically, we find this approach often leads to degraded model accuracy when the number of machines changes dramatically. Based on our theoretical analysis, the noise in the momentum state scales inversely with the mini-batch size. Linearly scaling the learning rate when increasing mini-batch size fails to consider the difference of variance noise scale, which leads to the optimization difficulty.

In this paper, we propose a new optimization strategy called *Dynamic Mini-batch Stochastic Gradient Descent (Dynamic SGD)* that smoothly adjusts the learning rate to stabilize the variance changes. We design multiple dynamic training environment settings by varying the number of GPUs between 8 and 128. We evaluate our proposed method on image classification (ResNet-50 [13], MobileNet [15] and InceptionV3 [42] on ImageNet [6]), object detection (SSD [26] on MS-COCO [25]), and semantic segmentation (FCN [27] on ADE20K [53]). The experiment results demonstrate that the convergence of Dynamic SGD in dynamic environments consistently matches the convergence with training in static environments.

The contributions of this paper include:

1. Identifying the key challenge to extend synchronized SGD with momentum into dynamic training environments.

2. Proposing a new method which we call Dynamic SGD to smoothly adjust the learning rate when the number of machines changes to stabilize the training.

3. Extensive empirical studies to benchmark the proposed Dynamic SGD method with straightforward approaches on large-scale image classification, object detection and semantic segmentation tasks.

2. Background

2.1. Mini-batch SGD with Momentum

We first review mini-batch stochastic gradient descent (SGD), and SGD with momentum update [38]. Given a network parameterized by vector w and a labeled dataset X, the loss to minimize can be written in the following form [11]:

$$L(w) = \frac{1}{|X|}\sum_{i=1}^{|X|} l(w, x_i), \tag{1}$$

where $l(w, x_i)$ is the loss for the sample x_i with parameters w. SGD iteratively updates parameters w with its gradient estimated within each mini-batch to optimize the loss:

$$w_{t+1} = w_t - \eta\frac{1}{B}\sum_{i=1}^{B} \nabla l(w_t, x_i), \tag{2}$$

where w_t is the network parameter at iteration t, η is the learning rate, and B is the number of samples randomly drawn from dataset X in a mini-batch.

In practice, SGD with momentum helps accelerate the optimization. The momentum state keeps the exponentially weighted past gradient estimates and updates the loss with the following rule:

$$\begin{aligned} u_{t+1} &= \mu u_t + \frac{1}{B}\sum_{i=1}^{B} \nabla l(w_t, x_i) \\ w_{t+1} &= w_t - \eta u_{t+1}, \end{aligned} \tag{3}$$

where u_t is the momentum state of historical gradients at iteration t, and μ is the decay ratio for the momentum state. The parameter update depends on both gradient estimated within current mini-batch and exponentially weighted gradients from historical mini-batches.

2.2. Data Parallel Distributed Training

In distributed training, multiple workers work together to finish a training job. A worker is a computational unit, such as a CPU or a GPU. Data parallelism defines how training workloads are partitioned into each worker.

Assume we are using mini-batch SGD with a mini-batch size B on N workers, and each worker has a copy of model parameters. In data parallelism, we partition a mini-batch into N parts, so each worker will get B/N examples, and

Algorithm 1 Data parallel distributed training for a single mini-batch.

Require: Mini-batch size B, number of workers N, and learning rate η
  Randomly sample B inputs $x_1, \ldots, x_B$
  Partition inputs into N parts $X_1, \ldots, X_N$
  for worker $i = 1, \ldots, N$ **do** ▷ Run in parallel
    Compute gradient ∇l_i based on data partition X_i
    Allreduce gradient by $\nabla l \leftarrow \sum_{i=1}^{N} \nabla l_i$
    Update parameters by $w \leftarrow w - \frac{\eta}{B} \nabla l$
  end for

then compute its local gradients. All local gradients are averaged over workers through synchronous communication to obtain the global gradient for this mini-batch. Then this global gradient is used to update model parameters by the SGD update rule. Algorithm 1 illustrates how a single batch is computed.

3. Methods

In this section, we first describe the setup of elastic distributed training environment, and then study the optimization instability of momentum SGD in such environment and introduce Dynamic Mini-batch SGD to stabilize the training.

3.1. Elastic Distributed Training System

In a dynamic scheduling deep learning system, the computation resources are managed, planned and distributed dynamically for different training tasks based on their priorities and system availability. We refer to the distributed training environment with dynamical computation resources as *Elatsic Distributed Training*.

An overview of elastic distributed training diagram for an example task is shown in Figure 2 (note that the system typically has more than one training tasks). The scheduler maintains the training state and coordinates the resources. It tracks the current and future number of workers by monitoring heartbeats from existing workers and availability notices. Each worker computes the gradients for the assigned data. The parameter server for the training task maintains the primary copy of model parameters, updates the parameters based on the aggregated gradients from workers, and send the updated parameters back to workers. In practice, the parameter server can partition the parameters among multiple machines to increase throughput.

3.2. Dynamic Mini-batch SGD

Learning Rate Scaling. Prior work linearly scales the learning rate according to the mini-batch size, which achieves success in large mini-batch training [11]. Considering the SGD without momentum update in Equation 2, the parameter update for k iterations using mini-batch size of B

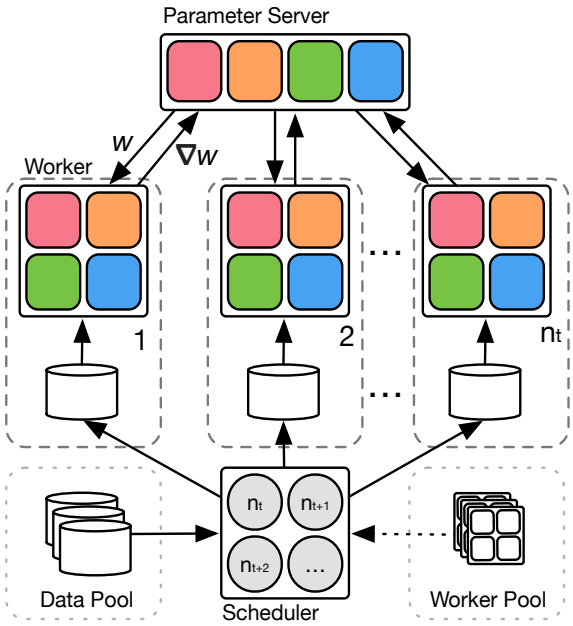


Figure 2: Overview of the elastic distributed training system with an example task. The scheduler keeps track of the training progress, monitors the worker pool, and dynamically assigns the data and workers to each task based on their priority and system availability. The task parameter server aggregates the parameter gradient ∇w from each worker and send back the updated parameter weights w. (n_t, n_{t+1}, and n_{t+2} represent the number of workers assigned to the current task at time step t, $t+1$ and $t+2$.)

can be approximated by updating once with linearly scaled learning rate $k\eta$ on the combined k mini-batches with the size of kB, if we could assume $l(w_t, x) \approx l(w_{t+j}, x) \forall j < k$. We discuss in further detail in the Appendix B.1. We refer to this strategy of linear scaling up the learning rate as *linear scaling*. Despite its success in large mini-batch training, linear scaling fails to apply to an elastic training environment, because the effect of momentum is not compensated. To our knowledge, the effect of momentum for changing mini-batch size has not been studied in the previous work.

SGD Momentum with Changing Mini-batch Size. To clearly see the weight update in momentum SGD as in Equation 3, a common way to rewrite the equation is substituting ηu_t with v_t to absorb the learning rate η [11]:

$$v_{t+1} = \mu v_t + \eta \frac{1}{B} \sum_{i=1}^{B} \nabla l(w_t, x_i)$$
$$w_{t+1} = w_t - v_{t+1}, \tag{4}$$

which is identical to Equation 3 for a static learning rate

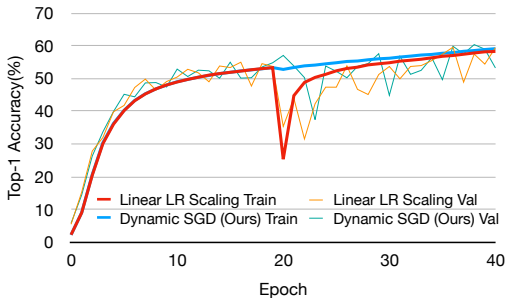


Figure 3: Training and validation accuracy on ImageNet. Increasing the mini-batch size from 1K to 12K at epoch 20. The proposed Dynamic SGD is robust to the mini-batch change, while the training with linear scaling get degraded.

η and mini-batch size B. For elastic training at iteration $t+1$, the mini-batch size is changed from B to kB (k is the changing ratio, which can be decimal). Applying the linear scaling directly for learning rate to the momentum SGD, we can get the paramter update as:

$$v_{t+1} = \mu k v_t^B + k\eta \frac{1}{kB} \sum_{i=1}^{kB} \nabla l(w_t, x_i)$$
$$w_{t+1} = w_t - v_{t+1}, \tag{5}$$

where v_t^B is the momentum state estimated on previous mini-batch size B, which is scaled to k times for momentum correction to maintain the equivalence with Equation 3 [11]. However, this compensation only considers the gradient scale[1] instead of the noise scale in the momentum state.

Noise Scale in the Gradient and Momentum State. SGD obtains an estimated gradient within a mini-batch of size B with $\nabla l^B(w) = \frac{1}{B}\sum_{i=1}^{B} \nabla l(w, x_i)$ to approximate the full gradient on the entire dataset $\nabla L(w)$. The gradient of each sample $\nabla l(w, x_i)$ is a random variable whose expected value is $\nabla L(w)$. The variance of estimated gradient within a mini-batch scales inversely to the mini-batch size B [29][2]. The gradient in a mini-batch gives a noisy estimate of the full gradient, and larger mini-batch size provides less noisy estimate.

The variance of the momentum state is proportional to the variance of the mini-batch gradient, *i.e.* $var(u_t) = \frac{1}{1-\mu^2} var(\nabla l^B(w))$. Therefore, directly scaling up the momentum state on the mini-batch size B for "momentum correction" as in Equation 5 increases the noise scale quadratically to k^2 times of the expected momentum state on mini-batch size kB. We also observe unstable training under such setting in the experiment, and the training curves are shown in Figure 7.

[1]The scale of the momentum state is $\frac{1}{1-\mu}$ times of the gradient scale.

[2]Assuming x_i is sampled independently from the dataset and $B \ll |X|$.

Momentum Compensation. To address the difficulty of adapting previous momentum state, we introduce momentum compensation factor γ_t, which gradually changes over iterations to allow smooth adaptation of the momentum state when increasing mini-batch size to k times. The weight update is given by $w_{t+1} = w_t - \gamma_{t+1}\eta u_{t+1}$, where γ_{t+1} is:

$$\gamma_t = \begin{cases} 1 + \dfrac{t - t_0}{T}(k-1) & \text{if } (t - t_0) < T \\ k & \text{otherwise} \end{cases} \tag{6}$$

t_0 is the iteration index when the mini-batch size is changed, and T is the total compensation iteration number, which we find $T = 8k$ works well[3]. We refer to the mini-batch SGD adopting momentum compensation strategy for elastic distributed training as *Dynamic Mini-batch Stochastic Gradient Descent (Dynamic SGD)*[4]. The proposed Dynamic SGD stabilizes training as shown in Figure 7.

4. Related Work

4.1. Large Mini-batch Distributed Training

Prior work achieves great success in large mini-batch data parallel training for deep convolutional neural networks. Li [23] shows distributed training with up to 5K mini-batch size without a loss in accuracy on ImageNet. Goyal *et al.* [12] employs linear learning rate scaling rule with warm-up scheme to train ResNet-50 for image classification with large batch size up to 8K without reducing accuracy. Layer-wise Adaptive Rate Scaling (LARS) optimization algorithm overcomes the optimization difficulties of larger batch training beyond 8K batch size, and scales ResNet-50 training up to 32K mini-batch. The mini-batch size is further increased to 64K using mixed precision training [47, 18]. Square root learning rate scaling and longer training time for is also proposed for training a large mini-batch size [14]. Despite their success in large mini-batch training, network training using dynamic mini-batch size is rarely studied.

4.2. Stochastic Gradient Descent

Asynchronous stochastic gradient descent (asyncSGD) [5] assumes machines are loosely coupled and thus suits this dynamic machine environment. Previous study demonstrates that it is difficult for asyncSGD to match the model accuracy using synchronized SGD with similar computation cost [5, 2]. Therefore we use synchronized SGD in this work instead of async SGD to achieve better model accuracy.

[3]We only compensate for momentum adaption when increasing mini-batch size, because reducing noise scale will not influence the training (as shown in Figure 4).

[4]We discuss other potential momentum compensation methods in the Future Work Section 6.

Our work also benefits from pioneering studies on learning rate and mini-batch size. McCandlish *et al.* [29] analyzes largest useful mini-batch size based on gradient noise scale. Prior work proposes to increase the mini-batch size instead of decaying the learning rate during the training results in less than 1% loss in accuracy on ImageNet when scaling learning rate up to 3 times [40, 7]. Jastrzębski *et al.* [17] shows learning rate schedules can be replaced with batch size schedules from theoretical analysis. Despite changing mini-batch sizes are used in these work, reserved computation resources are required due to fixed resource schedule instead of dynamically planned.

5. Experimental Results

In this section, we conduct a comprehensive benchmark of proposed Dynamic SGD and baseline approaches on image classification, object detection and semantic segmentation. For image classification, we compare Dynamic SGD with static baseline and linear scaling for state-of-the-art network architectures ResNet-50 [13], MobileNet [15] and the InceptionV3 [42] on ImageNet [6]. Then we go beyond image classification task and evaluate the proposed method for object detection using Single Shot multi-box Object Detector (SSD) [26] on MS-COCO dataset [25], and semantic segmentation using Fully Convolutional Networks (FCN) [27] on ADE20K [53].

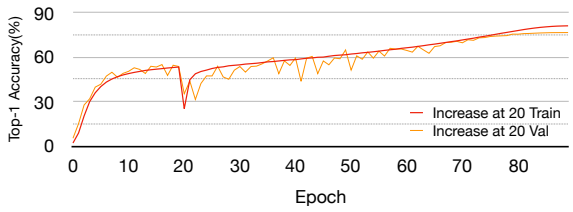
Baseline Approaches. In this experiment, we mainly compare the proposed Dynamic SGD with the following baselines:

- *Static Baseline*: no elasticity. The number of workers is fixed during the training.
- *Fixed Mini-batch Size*: fix the mini-batch size and (re)distribute the workload evenly to available workers.
- *Linear Scaling*: linearly scale up the mini-batch size and learning rate based on number of available workers.

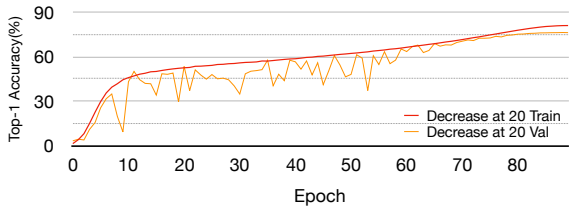
5.1. Image Classification

We first briefly describe the implementation details of the baseline network and the elastic training simulation. Then we compare the Dynamic SGD method with baseline approaches with suddenly increased number of workers using ResNet-50. Finally, we conduct a comprehensive study on state-of-the-art image classification models using randomly changing number of GPUs.

Implementation Detail. We adopt ResNet-50 [13], MobileNet 1.0 [15] and Inception V3 [42] as the baseline models and evaluate the performance on ImageNet-2012 [6] dataset.



(a) Increase the mini-batch size to 12 times at epoch 20.



(b) Decrease the mini-batch size to $\frac{1}{12}$ at epoch 20.

Figure 4: Top-1 accuracy on ImageNet validation set using ResNet-50. Influence of sudden change in mini-batch size using linear scaling approach. We use the 1K as the base mini-batch size. We find that the network training is relatively sensitive to increasing mini-batch size, but robust to decreasing.

The model is implemented in GluonCV[5] with MXNet [4]. Each network is trained for 90 epochs using cosine learning rate decay [28]. The learning rate is warmed-up for 5 epochs [11]. We use stochastic gradient descent (SGD) optimizer and set the momentum as 0.9 and weight decay as 0.0001. We use 8 GPUs with 128 per-GPU mini-batch size as the baseline, and set the base learning rate as 0.4. We linearly scale up the learning rate when increasing the mini-batch size. The input size is 224 by 224 for ResNet-50 and MobileNet 1.0, and 299 by 299 for Inception V3. We do not apply weight decay to biases as well as γ and β in batch normalization layers [18, 46], and will discuss its effect in the appendix. To study convergence when the mini-batch size changes, we simulate the gradient of a large mini-batch by accumulating gradients from multiple small mini-batches before applying the parameter update. For throughput analysis, we run the training job with multiple physical machines.

Increasing *vs.* Decreasing Mini-batch Size Influence. First we study the influence of a sudden change of number of GPUs at epoch 20. We train the network with two configurations: 1) training starts from 8 GPUs and then increase the number of GPUs to 96 at epoch 20, and 2) training starts from 96 GPUs and then decrease the number of GPUs to 8 at epoch 20. In both configurations we scale up or down the learning rate linearly with the number of GPUs. The results

[5]https://github.com/dmlc/gluon-cv

Method	Scale	Epoch 20	Epoch 70
Static Baseline	1×	76.91	
Static Baseline	12×	76.34	
Fixed Mini-batch Size	12×	76.28	75.87
Linear Scaling	12×	76.50	76.54
Dynamic SGD (ours)	12×	76.81	76.81

Table 1: Performance from different methods when increasing the number of GPUs at early or late stage of the training process. Our method is usually not affected by the sudden increment and outperforms the other approaches. The fixed mini-batch size method has worse performance with a late stage change.

are included in Figure 4. Figure 4 shows that by increasing the number of GPUs to 96, the training curve has a sharp drop, indicating the training process is drastically disrupted at epoch 20. On the other hand, decreasing the number of GPUs has no visible effect.

Baseline Comparisons. We then focus on the scenario with increasing number of GPUs. The number of GPUs is increased from 8 to 96 at epoch 20 and epoch 70. Besides linear scaling, we test 1) fixing the mini-batch size when increasing GPUs, i.e. reduce the per-GPU batch size, and 2) incorporating the learning rate warm up after the change. The results are in Table 1.

Table 1 shows that our Dynamic SGD method has a consistent performance against the sudden change at both early and late stage. Fixing the mini-batch size gives interesting results: it has a reasonable accuracy if the sudden increase happens at epoch 20, and gets worse if happens at epoch 70. When keeping the mini-batch size and increasing the number of GPUs, the per-GPU batch size is going to be a smaller number, which may affect the behavior of batch normalization layers [6]. Increasing at epoch 20 offers the network a longer time to train the batch norm layers thus results in a better accuracy.

Mini-batch Size Range. We compare our method with static baselines. By setting mini-batch size at 1K as the reference, Figure 5 includes the static baselines with larger mini-batch sizes, and our method with the mini-batch size increased at epoch 20. The static baseline with mini-batch size at 4K is slightly better than the baseline at 1K, which is likely to be caused by random variation. Starting from 8K the baseline keeps getting worse, while the Dynamic SGD method keeps up a better accuracy. It means that our method preserves the better initial status when training with a smaller mini-batch size, and can successfully transfer it

[6]We discuss batch normalization in the appendix.

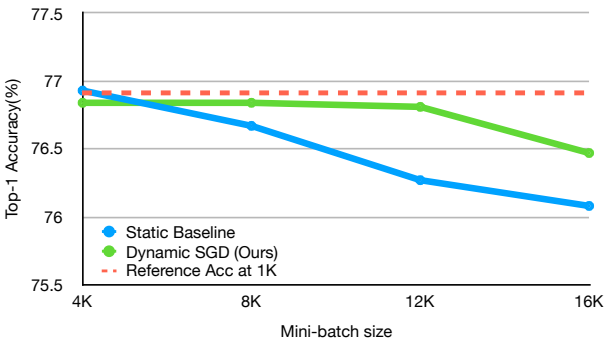


Figure 5: Top-1 validation accuracy *vs.* mini-batch size for Dynamic SGD and static baseline. The number of GPUs is increased at epoch 20. Dotted line is the performance of static baseline at 1K. Our method is close to the 1K reference and is often better than the static baseline at large mini-batch size.

Method	Scale	Throughput
Static Baseline	1×	4944
Fixed Mini-batch Size	12×	335
Dynamic SGD (ours)	12×	40095

Table 2: The training throughput (samples/sec) when increasing the number of GPUs from 8 to 96 at epoch 20. The static baseline uses a fixed number of GPUs of 8. The training throughput increases to 8.11 times when the number of GPUs increases by 12 times with Dynamic SGD method. For the fixed mini-batch size method, we observe significant slowdown when the number of GPUs increases.

to the training with larger mini-batch size. Our method has similar performance from mini-batch size at 4K to 12K, and it gets worse at 16K. This may be that the assumption $l(w_t, x) \approx l(w_{t+j}, x)$ in Section 3.2 does not hold anymore.

Throughput Analysis. We benchmark the training throughput of both fixed mini-batch method and the Dynamic SGD method. We use 12 machines, each equipped with 8 V100 GPUs and 2 Intel Xeon E5-2686 CPUs. The network bandwidth between machines is 25 Gbps. Table 2 shows the training throughput at epoch 20 when the number of GPUs increases to 12 times with the fixed mini-batch method and Dynamic SGD. The static baseline with 8 GPUs has a throughput of 4944 images per second.

When the number of GPUs increases in the middle of the training, these two methods exhibit different scaling property. For the fixed mini-batch size method, the training throughput drops to 335 images per second, about 7% of the static baseline throughput. Due to a smaller mini-batch size per GPU, less time is spent on computing the loss and

Method	Max Scale	ResNet-50	MobileNet 1.0	Inception V3
Static Baseline	1×	76.91	72.16	77.78
Static Baseline	12×	76.34	72.02	77.02
Static Baseline	16×	76.08	71.82	76.30
Linear Scaling	12×	76.80 (± 0.50)	72.12	77.72
Dynamic SGD (ours)	12×	76.88 (± 0.26)	72.27	77.89
Linear Scaling	16×	76.10 (± 0.33)	71.56	76.88
Dynamic SGD (ours)	16×	76.65 (± 0.56)	71.99	77.61

Table 3: Top-1 accuracy for three models on ImageNet 2012 validation set with randomized GPU configurations. Static Baselines get worse with larger mini-batch size. When the maximum scale of mini-batch size is 12 times, both linear scaling and Dynamic SGD work well. When the maximum scale is pushed to 16 times, linear scaling gets much less accurate while Dynamic SGD keeps a better performance.

gradient, and more time is spent on communication and synchronization between GPUs. In contrast, the Dynamic SGD method increases the mini-batch size as the number of GPUs increases, which leads to a higher computation-communication ratio. Increasing the number of GPUs to 12 times boosts the training throughput to 8.11 times.

Benchmark on Random Schedules. We further study a more general scenario in which the number of GPUs may change frequently to a random scale up to a certain limit. We simulate the environment with two configurations, where the minimum number of GPUs is 8, and the maximum number of GPUs is 96 (12 times) and 128 (16 times) respectively. Given the range, we randomly change the number of GPUs within the range every 5 epochs. We have performed experiments for three image classification networks: ResNet-50, MobileNet 1.0 and Inception V3 with an identical training setup. To have more robust results, we repeatedly generate the random schedules on ResNet-50 for 5 times and report the mean and standard deviation of the top-1 accuracy metric. All results are included in Table 3.

In Table 3, we fix the numbers of GPUs to 8, 96 and 128 during the training as baselines. For all three networks, the baselines with 96 GPUs loose accuracies, and with 128 GPUs drop accuracies significantly. It can also be seen that MobileNet 1.0 is more robust to the number of GPUs than the other two networks.

For each experiment for the random schedules, we randomly generate a schedule for the number of GPUs and train both our method and the linear scaling method with it, so that the results are comparable. When the maximum number of GPUs is 96, we can see that both our method and linear scale are better than the baseline with 96 GPUs and they have similar performance. However if we further set the maximum number of GPUs to 128, we can see that our method is still close to our 8-GPU baseline while the linear scale method has a much worse performance.

Based on the experimental results, our method has a consistently good performance and is comparable to the 8-GPU baseline accuracy. The linear scaling method can keep the performance if the maximum number of GPUs is not too large, e.g. twelve times of the baseline in our experiments, and its performance drops if we push the maximum number further.

5.2. Elastic Training for Object Detection

We go beyond the image classification and study elastic training on the object detection task. We first describe the implementation detail of the baseline network and the simulation procedure. We then study the performance with random schedules.

Implementation Detail We train Single Shot Multi-box Object Detector network (SSD) [26] on the MS-COCO dataset [25]. We use *train2017* imageset for training, and *val2017* imageset for validation. We adopt ResNet-50 as the backbone and use 512×512 as the input image size. We use extensive data augmentation, hard negative sampling mining and normalize the loss by valid object count[26]. The baseline network is trained with 0.008 learning rate, mini-batch size 256 on 32 GPUs, with a cosine learning rate schedule and gradual learning rate warmup in the first 5 epochs[50].

We randomly generate a schedule for the number of GPUs and train both our method and linear scaling with it. The number of GPUs changes every 5 epochs with a maximum scale of 12 times and 16 times respectively. When the maximum scale is 12 times, we see that both linear scaling method and ours performs relatively good compared to the static baseline. If the maximum scale is set to 16 times, we can see that our method is still close to our static baseline, while the linear scaling method fails to converge to a reasonable accuracy.

Method	Max Scale	mAP
Static Baseline	1×	30.3
Static Baseline	12×	30.0
Static Baseline	16×	29.6
Linear Scaling	12×	30.0
Dynamic SGD (ours)	12×	30.1
Linear Scaling	16×	20.2
Dynamic SGD (ours)	16×	29.6

Table 4: Mean average precision (mAP) for SSD network on the MS-COCO 2017 val set with randomized GPU configurations. with randomized GPU configurations. We report mAP with IoU threshold 0.5:0.95, 0.5 and 0.75. When the maximum scale of mini-batch is 12 times, both linear scaling and Dynamic SGD perform relatively well. However, linear scaling method breaks down when the maximum scale is set to 16 times, while ours has similar performance compared to static baselines.

5.3. Elastic Training for Semantic Segmentation

We also evaluate the Dynamic SGD method on semantic segmentation on ADE20K dataset [53], which is a large scale scene parsing benchmark containing 20K/2K/3K images for training/validation and test set. We use Fully Convolutional Network (FCN) [27] as the baseline approach. Following the prior work [51], we use the ResNet-50 [13] as the base network and apply the dilation strategy at stage 3 & 4, resulting in a stride-8 model. We use synchronized Batch Normalization[7] [48] with a working batch size of 16. The baseline model is trained using a mini-batch size of 16, and base learning rate of 0.1 with cosine learning rate decay. For random schedule training, the training starts with 8 GPUs and the GPU scale is randomly chosen from 8 to maximum scales (12 times, 16 times) every 5 epochs.

Pixel accuracy (pixAcc) and mean intersection of union (mIoU) are used as the evaluation metrics. The results are shown in Table 5. The static baseline 1× achieves 78.99% pixAcc and 39.48% mIoU. For random schedule, the proposed Dynamic SGD always outperform linear scaling method and achieves comparable performance with static baseline 1×. For static baseline 12× and static baseline 16×, FCN gets dramatically worse performance, which we believe is because the running statistics in batch normalization is not compensated for large mini-batch size. Different from SSD experiment where the batch normalization uses fixed batch statistics, FCN training employs Synchronized Batch Normalization. The moving averages of batch statistics are accumulated with momentum update, which is much slower than the update of network weights for large mini-batch size. This leads to degraded performance for static baseline. For the experiments using random schedule, the mini-batch size may not be always large and may alleviate the problem. Image classification training is not impacted by the batch normalization, because the training has larger number interactions per batch to accumulate batch statistics.

[7]SyncBN is described in the appendix.

Method	Max Scale	pixAcc %	mIoU %
Static Baseline	1×	78.99	39.48
Static Baseline	12×	76.76	34.69
Static Baseline	16×	76.01	33.65
Linear Scaling	12×	78.94	39.16
Dynamic SGD (ours)	12×	79.07	39.34
Linear Scaling	16×	79.01	38.90
Dynamic SGD (ours)	16×	79.02	39.34

Table 5: Elastic Training results using random schedules on semantic segmentation, showing pixAcc and mIoU of FCN on ADE20K validation set. The Dynamic SGD is compared with static baseline and linear scaling using 12× and 16× maximum GPU scales.

6. Conclusion and Future Work

In this paper, we study the optimization difficulty in elastic distributed training, which is the fundamental bottleneck for dynamic scheduling in deep learning system. We find this difficulty is mainly because the momentum is not compensated for the changing mini-batch size. For this, we introduce Dynamic SGD to gradually adapt the momentum. The proposed Dynamic SGD has been evaluated on three major computer vision tasks: image classification, object detection and semantic segmentation. The experimental results demonstrate the proposed method stabilizes the training and achieves comparable model accuracy after convergence with single node training using the same training settings. The proposed Dynamic SGD method can also employ other momentum compensation strategies, which is discussed in the appendix.

References

[1] Z. Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *The Journal of Machine Learning Research*, 18(1):8194–8244, 2017.

[2] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[4] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and effi-

cient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[7] A. Devarakonda, M. Naumov, and M. Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017.

[8] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition. *arXiv preprint arXiv:1812.03982*, 2018.

[9] S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[10] Google. Preemptible vm, Jan 2018.

[11] 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. *arXiv preprint arXiv:1706.02677*, 2017.

[12] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

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

[14] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] S. Jastrzębski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017.

[18] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[19] C. Jin, P. Netrapalli, and M. I. Jordan. Accelerated gradient descent escapes saddle points faster than gradient descent. *arXiv preprint arXiv:1711.10456*, 2017.

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

[21] J. Konečnỳ, J. Liu, P. Richtárik, and M. Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2016.

[22] J. Lei, L. Yu, M. Bansal, and T. L. Berg. Tvqa: Localized, compositional video question answering. *arXiv preprint arXiv:1809.01696*, 2018.

[23] M. Li. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, PhD thesis, Carnegie Mellon University, 2017.

[24] Y. Li, J. Shi, and D. Lin. Low-latency video semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5997–6005, 2018.

[25] 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 *European conference on computer vision*, pages 740–755. Springer, 2014.

[26] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[28] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.

[29] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. An empirical model of large-batch training. *CoRR*, abs/1812.06162, 2018.

[30] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[31] Microsoft. Azure low priority vm, April 2018.

[32] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[33] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[34] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[35] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[36] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. J. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015.

[37] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

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

[39] A. W. Services. Amazon ec2 spot instances, August 2018.

[40] S. L. Smith, P.-J. Kindermans, and Q. V. Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[43] M. Tapaswi, Y. Zhu, R. Stiefelhagen, A. Torralba, R. Urtasun, and S. Fidler. Movieqa: Understanding stories in movies through question-answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4631–4640, 2016.

[44] P. Voigtlaender, Y. Chai, F. Schroff, H. Adam, B. Leibe, and L.-C. Chen. Feelvos: Fast end-to-end embedding learning for video object segmentation. *arXiv preprint arXiv:1902.09513*, 2019.

[45] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[46] J. Xie, T. He, Z. Zhang, H. Zhang, Z. Zhang, and M. Li. Bag of tricks for image classification with convolutional neural networks. *arXiv preprint arXiv:1812.01187*, 2018.

[47] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. arxiv preprint. *arXiv preprint arXiv:1708.03888*, 2017.

[48] H. Zhang, K. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal. Context encoding for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[49] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. *arXiv preprint arXiv:1511.05950*, 2015.

[50] Z. Zhang, T. He, H. Zhang, Z. Zhang, J. Xie, and M. Li. Bag of freebies for training object detection neural networks. *arXiv preprint arXiv:1902.04103*, 2019.

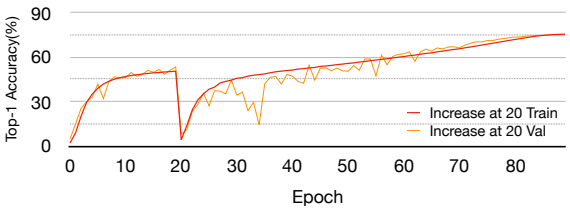
[51] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 6230–6239. IEEE, 2017.

[52] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4120–4129. JMLR. org, 2017.

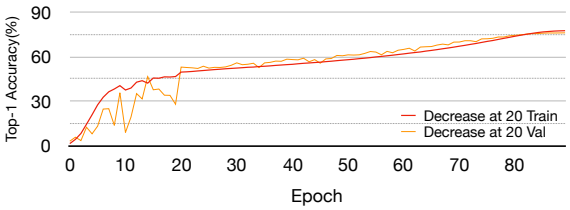
[53] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Method	Scale	Epoch 20	Epoch 70
Static Baseline	1×	76.89	
Static Baseline	12×	76.27	
Linear Scaling	12×	75.03	76.43
Dynamic SGD (ours)	12×	76.47	76.78

Table 6: Performance of different methods when increasing the number of GPUs at early or late stage of the training process, when weight decay is present. Our method is usually not affected by the sudden increment and outperforms the linear method. Linear scaling breaks down if the change happens at epoch 20, 1.86% worse than the static 1 × baseline.



(a) Increase the mini-batch size to 12 times at epoch 20.



(b) Decrease the mini-batch size to $\frac{1}{12}$ at epoch 20.

Figure 6: Top-1 accuracy on ImageNet validation set using ResNet-50, when weight decay is applied. Influence of sudden change in mini-batch size using linear scaling approach. We use the 1K as the base mini-batch size. We find that the network training is relatively sensitive to increasing mini-batch size, but robust to decreasing.

Appendix

A. Linear Scaling with Increased #GPUs

Weight decay is a common practice to regularize neural networks. The experimental results at Section 5.1 do not apply weight decay biases as well as γ and β in batch normalization layers. In this section, we study the case where weight decay is applied. Here we also adopt ResNet-50 [13] and train it using the same implementation as described in Section 5.1. The only difference is that here weight decay is applied to biases as well as γ and β in batch normalization layers. We also simulate the gradient of a large mini-batch by accumulating gradients from multiple small mini-batches before applying the parameter update.

We study the influence of a sudden change of number of GPUs at epoch 20. We train the network with two configurations: 1) training starts from 8 GPUs and then increase the number of GPUs to 96 at epoch 20, and 2) training starts from 96 GPUs and then decrease the number of GPUs to 8 at epoch 20. In both configurations we scale up or down the learning rate linearly with the number of GPUs. The results are included in Figure 6.

Figure 6 shows that by increasing the number of GPUs to 96, the training curve has a sharp drop, indicating the training process is drastically disrupted at epoch 20. On the other hand, decreasing the number of GPUs has no visible effect. This leads to a more through study for the case where the number of GPUs is suddenly increased.

Table 6 shows that our Dynamic SGD method has a consistent performance against the sudden change at both early and late stage. The linear scaling method of learning rate works well if we increase the number of GPUs at a later epoch. If the increase happens at epoch 20, the linear scaling method perform much worse than the static baseline 12 ×. Meanwhile our method performs consistently well on increasing number of GPUs at both early and later epochs.

B. Discussions

This section provides further theoretical discussions on (1) our choice of linear relations between the learning rate and the number of machines and (2) our decision to warm-up linear rate when the number of machines changes. The discussions extend the main text in terms of technical details and generality.

B.1. Linear LR Scaling in Convex Optimization

We briefly mention the reasons with a simplified convex problem, similar to [30]. First, optimal learning rate in gradient descent is chosen by optimizing a conservative upper bound,

$$L(w_t - \eta_{t+1}\nabla l(w_t)) \leq L(w_t) - \eta_{t+1}\nabla l(w_t)^\top \nabla L(w_t) + \frac{C\eta_{t+1}^2}{2}\nabla l(w_t)^\top \nabla l(w_t),$$

where C is the Lipschitz constant which bounds all smaller terms in the Taylor expansion. If we additionally assume an i.i.d. Gaussian noise of the stochastic gradient,

$$\nabla l(w_t) = \nabla L(w_t) + \varepsilon_t,$$

where $\varepsilon_t \sim \mathcal{N}(0, \sigma_k^2)$ is the noise of the average gradient estimation. Here σ_k^2 is the variance of ε from k machines, which decreases as k increases, i.e., $\sigma_k^2 = \frac{\sigma_1^2}{k}$. The expected

loss becomes,

$$\mathbb{E}L(w_t - \eta_{t+1}\nabla l(w_t)) \leq L(w_t) - \eta_{t+1}\nabla L(w_t)^\top \nabla L(w_t) + \frac{C\eta_{t+1}^2}{2}(\nabla L(w_t)^\top \nabla L(w_t) + \sigma_k^2).$$

Define $G^2 = \nabla L(w_t)^\top \nabla L(w_t)$, the optimal choice is $\eta_{t+1} = \frac{1}{C}\frac{G^2}{G^2+\sigma_k^2} = \frac{kG^2}{C(kG^2+\sigma_1^2)} \approx \frac{kG^2}{C\sigma_1^2}$,where the last approximation assumes $kG^2 \ll \sigma_1^2$, common in SGD with many mini-batches. In this way, we show that η_{t+1} grows linearly with machine size k.

B.2. Linear LR Scaling in Non-convex Optimization

This section extends the previous discussions to non-convex stochastic optimization. Our theoretical analysis also extends [9], which showed that non-convex stochastic gradient descent converges to a stationary point at a near-optimal rate of $O(1/\sqrt{T})$.

Assumption 1. *Let $L_\Delta = L(w_0) - \min L(w) < \infty$ be the total range of the objective function, which is, without loss of generality, finite. Suppose $L(w)$ has bounded second-order gradient, $-CI \preceq \nabla^2 f(w) \preceq CI$. For iteration t, let the number of machines be k_t such that $1 \leq k_t \leq K$, where K is the maximum machine size, and the variance of the stochastic gradient $\nabla l_t(w)$ be $\mathbb{E}[\|\nabla l_t(w) - \nabla L_t(w)\|^2] \leq \frac{\sigma_1^2}{k_t}$, which is the same assumption we used in the appendix of the main text.*

We aim to analyze the convergence of a dynamic learning rate scaling rule, where at iteration t, the step size is $\eta_t = k_t^\beta \eta_0$ for some constant $\beta \geq 0$. Choosing $\beta = 0$ yields a constant learning rate, which may be suboptimal. Define constants $T_0 = \frac{2CKL_\Delta}{\sigma_1^2}$, $C_1 = \sqrt{\frac{2L_\Delta}{C\sigma_1^2}}$ and $C_2 = \sqrt{2C\sigma_1^2 L_\Delta}$, which do not depend on the choice of the step sizes η_t.

Theorem 2 (Convergence with Dynamic Machine Size)**.** *Under Assumption 1 with dynamic machine sizes $k_t \forall t$, if we adopt a strategy that sets the step size to $\eta_t = k_t^\beta \eta_0$, where $\beta \geq 0$ is a predefined constant, and have sufficiently many gradient steps, $T(> T_0)$, then choosing*

$$\eta_t = \frac{C_1 k_t^\beta}{\sqrt{\sum_{t=1}^T k_t^{2\beta-1}}} \tag{7}$$

guarantees at least one solution at the stationary point whose expected gradient is at most:

$$\min_{t \leq T} \mathbb{E}[\|\nabla L(w_t)\|^2] \leq C_2 \frac{\sqrt{\sum_{t=1}^T k_t^{2\beta-1}}}{\sum_{t=1}^T k_t^\beta}. \tag{8}$$

Remark 1. *Equation* (8) *converges to zero for any learning rate scaling rule $\eta_t = k_t^\beta \eta_0$, as long as $\beta > 0$ and Equation* (7) *is satisfied. For example, with a fixed step size $\eta_t = \eta_0 (< \frac{1}{C})$, Theorem 2 reduces to [32].*

Remark 2. *Equation* (8) *suggests that the upper bound of the convergence rate becomes optimal when the learning rate is $\eta_t = k_t \eta_0$ with $\beta = 1$. To see how, notice that the right-hand side of* (8) *can be relaxed using Cauchy's inequality as*

$$\frac{\sqrt{\sum_{t=1}^T k_t^{2\beta-1}}}{\sum_{t=1}^T k_t^\beta} \geq \frac{1}{\sqrt{\sum_{t=1}^T k_t}}, \tag{9}$$

which is tight when $\beta = 1$. The convergence rate is comparable with $O(1/\sqrt{T})$ when the machine size is fixed.

Remark 3. *In practice, the number of steps T and the bound on the second-order gradients C may not be known ahead of time. To adapt to any C and T, the learning rate is often also decreasing over time, i.e. $\eta_t = f(t)k_t^\beta \eta_0$. In our experiments, the learning rate follows a cosine function $f(t) = \cos(\frac{\pi}{2}\frac{t}{90})$, which is monotonically decreasing over time $(t < 90)$. Theoretical analysis [32] on a similar decreasing rate $f(t) = 1/t$ shows that it has a suboptimal convergence rate of $O(1/\log(T))$, but is more generalizable to unknown conditions.*

Proof of Theorem 2. The original proof in [9] focuses on the quantity

$$\mathcal{E} = \frac{1}{\sum_{t=1}^T \eta_t} \sum_{t=1}^T \eta_t \mathbb{E}[\|\nabla L(w_t)\|^2], \tag{10}$$

which is an upper bound of the left-hand side of (8), because the weighted average of a sample is greater than its minimum value. To bound $\mathcal{E}$, [9] then expands the noise condition in Assumption 1 to obtain

$$\mathcal{E} \leq \frac{2L_\Delta}{\sum_t^T \eta_t} + \frac{\sum_t^T C\eta_t^2 \sigma_t^2}{\sum_t^T \eta_t}, \tag{11}$$

where the first term connects the changes in the function values with noiseless gradients and the second term reflects the noise introduced by using stochastic gradients. In our case, the stochastic gradient noise at iteration t is $\sigma_t^2 \leq \frac{\sigma_1^2}{k_t}$. We further plug in the learning rate $\eta_t = k_t^\beta \eta_0$ to rewrite (11) as

$$\mathcal{E} \leq \frac{2L_\Delta}{\eta_0 \sum_t^T k_t^\beta} + \frac{\eta_0 C\sigma_1^2 \sum_t^T k_t^{2\beta-1}}{\sum_t^T k_t^\beta}. \tag{12}$$

Since (12) works for any choice of η_0, Cauchy's inequality suggests the optimal η_0 such that the two terms on the right-hand side equal, i.e., $\eta_0^2 = \frac{2L_\Delta}{C\sigma_1^2}\frac{1}{\sum_{t=1}^T k_t^{2\beta-1}} = \frac{C_1^2}{\sum_{t=1}^T k_t^{2\beta-1}}$. Straightforward calculation recovers (7) and (8). □

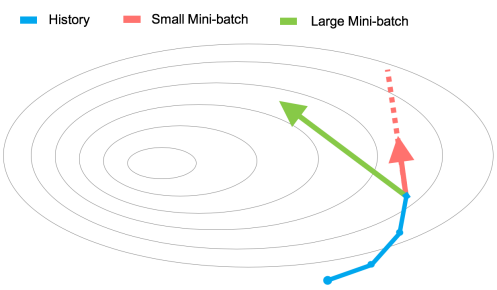


Figure 7: Gradient of larger mini-batch usually has better approximation of the true gradient. We assume the momentum state has a similar direction with gradient as shown using red dashed line. In momentum SGD, increasing the mini-batch size during the training without compensating for the history momentum.

B.3. Momentum and Learning Rate Warm-up

Momentum is helpful in convex problems with gradient descent by improving the condition number of the objective function [34, 33]. Momentum also helps in non-convex problems with gradient descent by escaping the saddle points [19]. While momentum in SGD can be prone to error accumulation [21], their adoption usually helps in practice, especially when the gradient noise is small in large batch training [1].

The challenge we face in a dynamic environment is that the variance of the momentum terms does not immediately change after the machine size changes. This violates Assumption 1 in Theorem 2 that the variance in the descent direction is inversely proportional to the machine size, e.g., in the case of SGD without momentum. However, the variance of the momentum will monotonically decrease until $O(1/1-\mu)$ number of epochs, where μ is the momentum decay rate, e.g., $\mu = 0.9$. We therefore apply a learning rate warm-up strategy such that the variance in the descent direction, which is the product of the variance of the momentum itself and the step size, maintains a relatively stable value. After the learning rate warm-up, the variance of the descent direction will also be inversely proportional to the machine size and the same linear scale would apply.

B.4. Thoughts on Momentum Compensation

We adopt a simple yet effective strategy in this paper to gradually increase the learning rate for smooth adaption of momentum state, which works well empirically. However, the proposed Dynamic SGD can also employ other momentum compensation strategies. Inspired by the physical analogy of Newtonian particles in a conservative force field in Qian [35], we can consider the loss $L(w)$ as the energy function, μ as the system friction coefficient, and the v_t as the velocity in Equation 4 and the mini-batch gradient $\sum_{i=1}^{B} \nabla l(w_t, x_i)$ as the acceleration. Therefore, the "velocity" can be dynamically adjusted by the system without introducing extra hyper-parameters, such as momentum compensation factor γ_t. The parameter update is given by:

$$\begin{aligned} v_{t+1} &= \mu v_t + \hat{\eta} \sum_{i=1}^{B} \nabla l(w_t, x_i) \\ w_{t+1} &= w_t - v_{t+1}, \end{aligned} \tag{13}$$

where $\hat{\eta}$ is the step size, which is different from learning rate as it is not coupled with mini-batch size. Comparing to the method proposed in Section 3.2, 1) this strategy does not introduce extra hyper-parameter, 2) does not need warm-up-like gradually adaption, 3) is generalized both increasing and decreasing mini-batch situations. The experimental analysis for different momentum compensation will be addressed in our future work.

C. Batch Normalization with Data Parallel

The batch normalization layer [16] normalizes the data within a mini-batch, which makes the network less sensitive to the initialization and allows larger learning rate training. Batch normalization is commonly used in modern deep convolution neural networks. Standard implementation of batch normalization in data parallel training normalizes the data within each worker or GPU.

Denotes the per-worker batch size as S, and the inputs to a batch normalization layer as $x_1, \ldots, x_S$. The mean μ and variance σ^2 for the inputs are given by $\mu = \frac{1}{S}\sum_{i=1}^{S} x_i$ and $\sigma^2 = \frac{1}{S}\sum_{i=1}^{S}(x_i - \mu)^2$. The outputs $y_i \in \{y_1, \ldots, y_S\}$ is given by:

$$y_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{14}$$

where γ and β are the scale and shift parameters, and ϵ is a small constant to avoid extreme values. As can be seen, when changing the per-worker batch size S, the batch normalization layer standardizes the inputs differently.

Different than the standard implementation of BN, Synchronized Batch Normalization (SyncBN) [48] normalizes the data using mean and variance calculated across multiple worker/GPUs, and synchronized their gradients during the backward pass. SyncBN is usually used in semantic segmentation task, where the per-worker batch size is often small. It is not suitable for image classification, because the synchronization becomes a big communication overhead to the training.