# Accelerated Large Batch Optimization of BERT Pretraining in 54 minutes

**Shuai Zheng, Haibin Lin, Sheng Zha, Mu Li**
{shzheng, haibilin, zhasheng, mli}@amazon.com
Amazon Web Services

## Abstract

BERT has recently attracted a lot of attention in natural language understanding (NLU) and achieved state-of-the-art results in various NLU tasks. However, its success requires large deep neural networks and huge amount of data, which result in long training time and impede development progress. Using stochastic gradient methods with large mini-batch has been advocated as an efficient tool to reduce the training time. Along this line of research, LAMB is a prominent example that reduces the training time of BERT from 3 days to 76 minutes on a TPUv3 Pod. In this paper, we propose an accelerated gradient method called LANS to improve the efficiency of using large mini-batches for training. As the learning rate is theoretically upper bounded by the inverse of the Lipschitz constant of the function, one cannot always reduce the number of optimization iterations by selecting a larger learning rate. In order to use larger mini-batch size without accuracy loss, we develop a new learning rate scheduler that overcomes the difficulty of using large learning rate. Using the proposed LANS method and the learning rate scheme, we scaled up the mini-batch sizes to 96K and 33K in phases 1 and 2 of BERT pretraining, respectively. It takes 54 minutes on 192 AWS EC2 P3dn.24xlarge instances to achieve a target F1 score of 90.5 or higher on SQuAD v1.1, achieving the fastest BERT training time in the cloud. A fast implementation of LANS is available online [1].

## 1 Introduction

Deep neural networks have achieved remarkable performance in various tasks such as image classification [13], speech recognition [10], machine translation [28], and natural language understanding [5]. These problems are typically formulated as the minimization of a nonconvex objective on a set of training samples. The most popular optimization tool is stochastic gradient descent (SGD) [22, 8, 2], which is simple and computationally efficient. However, deep learning thrives with large model size and huge amount of data, and it raises significant challenge even for a cheap optimizer such as SGD to reach a decent solution in a reasonable amount of time. For example, training a large BERT model requires 3 days on 16 TPUs [5] and it takes 40 days to train an AlphaGo Zero system [24]. Thus, it is necessary to develop fast optimization methods to accelerate deep neural network training.

To improve the training efficiency, ones often have to distribute the computation of a large mini-batch gradient to multiple computing nodes [3, 35]. Distributed synchronous SGD has become a de-facto method for large-scale machine learning problems. For further acceleration, variants that use classic momentum [20] and Nesterov's momentum [18] have been widely adopted [26, 13]. By increasing the mini-batch size, distributed synchronous SGD can make use of a larger learning rate so that the total number of training iteration can be reduced accordingly. The learning rate typically grows

[1] `https://github.com/szhengac/apex/blob/lans/apex/optimizers/fused_lans.py`

with the square root of the mini-batch size [4] or can even increase linearly with the mini-batch size when appropriate warmup schedule is employed [9]. However, one cannot increase the learning rate indefinitely and the learning rate scaling heuristics that depends on the mini-batch size can break for some cases [23]. Thus, it takes more efforts to search for good hyper-parameters for synchronous SGD methods .

To accelerate the convergence of SGD and spend less work in tuning hyper-parameters, many coordinate-wise adaptive learning rate based methods have been introduced [1, 7, 14, 27, 33, 21]. Adaptive gradient methods dynamically adjust their learning rates according to the received noisy gradients. On the other hand, several attempts have been made to use layer-wise learning rates for different layers [25, 29, 32, 34]. It has been shown that layer-wise learning rate improves generalization performance in practice. Very recently, LAMB is proposed in [30]. It combines AdamW optimizer [16] with normalized gradient descent [19, 12]. It is shown that LAMB managed to train BERT with a large mini-batch size of 64K without losing accuracy. However, it cannot further scale up to an even larger mini-batch size.

In this paper, we introduce per-block gradient normalization to LAMB and modify its momentum term by taking advantage of the connection between the classic momentum and Nesterov's momentum. The resultant accelerated gradient method is called LANS. Moreover, as the linear scaling only works up to certain mini-batch sizes, we propose to add a constant learning rate stage after the warmup phase. Such change allows the optimizer to use the maximum learning rate for a longer period of time, which results in sufficient training progress even we cannot further increase the maximum learning rate. The experimental results show that the proposed methods can use a very large mini-batch size of 96K and reduce the BERT pretraining time to 54 minutes on 192 Amazon EC2 P3dn.24xlarge instances without suffering from any performance deterioration.

**Notations**. For a vector $x \in \mathbb{R}^d$, $\sqrt{x}$ is the element-wise square root of $x$, $x^2$ is the coordinate-wise square of $x$, $\|x\|_2 = \sqrt{x^T x}$. For two vectors $x$ and $y$, $x/y$ denotes the element-wise division.

## 2 Related Works

In machine learning, one is interested in minimizing a $\ell_2$-norm regularized optimization problem of the form

$$\min_{x \in \mathbb{R}^d} F(x) = \mathbf{E}_\xi[f(x, \xi)] + \frac{\lambda}{2}\|x\|_2^2, \tag{1}$$

where $f$ is some possibly nonconvex loss function, $\xi$ is a random sample, $x$ is the model parameter, $\lambda$ is the regularization parameter, and the expectation is taken w.r.t. the underlying sample distribution. The objective (1) reduces to the expected risk that measures the generalization performance on unseen data [2] when $\lambda = 0$, and reduces to the regularized empirical risk when a finite training set is considered.

### 2.1 LAMB

For the gradient $g_t \in \mathbb{R}^d$, let $g_t = [g_{t,\mathcal{G}_1}, g_{t,\mathcal{G}_2}, \ldots, g_{t,\mathcal{G}_B}]$ be its decomposition into $B$ blocks, where $\mathcal{G}_b$ is the set of indices in block $b$, and $g_{t,\mathcal{G}_b}$ is the corresponding block of variables. A block can be a parameter tensor/matrix/vector. Recently, You *et al.* [30] introduced a layer-wise adaptive large batch optimization method, called LAMB (Algorithm 1). LAMB proposed to add a normalization factor to the AdamW (ADAM with weight decay) [16] update that divides the update by its $\ell_2$ norm. This ensures that the update for each block has unit $\ell_2$-norm. And, the learning rate is rescaled by $\phi(\|x_{t,\mathcal{G}_b}\|_2)$ for some function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$. In practice, it is generally set to an identity mapping. In this case, the update preserves the same $\ell_2$ norm as the model parameters, and the model parameters change in a smooth trajectory.

It was shown in [30] that LAMB enables a very large mini-batch size of 64K for training BERT while being able to achieve comparable accuracy to the small mini-batch size. With such large mini-batch size, the training time of BERT pretraining reduced from 3 days to 76 minutes on 1024 TPUs.

**Algorithm 1** LAMB [30]

**Input:** step size sequence $\eta_t$; $0 < \beta_1, \beta_2 < 1$; scaling function $\phi$; $\epsilon > 0$; regularization parameter $\lambda$.
**Initialize:** $x_1 \in \mathbb{R}^d$; $m_0, v_0 = 0$.
**for** $t = 1, \ldots, T$ **do**
  Compute mini-batch stochastic gradient $g_t$
  **for** $b = 1, 2, \ldots, B$ **do**
    $m_{t,\mathcal{G}_b} = \beta_1 m_{t-1,\mathcal{G}_b} + (1-\beta_1) g_{t,\mathcal{G}_b}$
    $v_{t,\mathcal{G}_b} = \beta_2 v_{t-1,\mathcal{G}_b} + (1-\beta_2) g^2_{t,\mathcal{G}_b}$
    $\tilde{m}_{t,\mathcal{G}_b} = m_{t,\mathcal{G}_b} / (1-\beta_1^t)$
    $\tilde{v}_{t,\mathcal{G}_b} = v_{t,\mathcal{G}_b} / (1-\beta_2^t)$
    compute ratio $r_{t,\mathcal{G}_b} = \frac{\tilde{m}_{t,\mathcal{G}_b}}{\sqrt{\tilde{v}_{t,\mathcal{G}_b}} + \epsilon}$
    $x_{t+1,\mathcal{G}_b} = x_{t,\mathcal{G}_b} - \eta_t \frac{\phi(\|x_{t,\mathcal{G}_b}\|_2)}{\|r_{t,\mathcal{G}_b} + \lambda x_{t,\mathcal{G}_b}\|} (r_{t,\mathcal{G}_b} + \lambda x_{t,\mathcal{G}_b})$
  **end for**
**end for**

### 2.2 Nesterov Momentum

Momentum methods have been widely used in training deep networks [26]. The classic momentum method, also known as heavy-ball method, introduced in [20] accumulates the past gradients $g_t$'s into a momentum vector $m_t$ (with $m_0 = 0$), which serves as a smoothing of the velocity:

$$m_t = \mu m_{t-1} + g_t \tag{2}$$
$$x_{t+1} = x_t - \eta_t m_t, \tag{3}$$

where $\mu \in [0, 1)$ is the momentum parameter. For a twice differentiable strongly convex function, it is known that the classic momentum method can be used to accelerate the gradient descent and improve the convergence rate from $O((1-\kappa)^t)$ to $O((1-\sqrt{\kappa})^t)$. where $\kappa$ is the condition number of the functions. For training deep neural network, the momentum method accelerates early optimization and helps gradient descent method escape from the local minimums.

Nesterov's accelerated gradient (NAG) [18, 26] is another kind of momentum method that is closely related to the classic momentum method in that it can be written as:

$$m_t = \mu m_{t-1} + g_t$$
$$x_{t+1} = x_t - \eta_t(\mu m_t + g_t).$$

Expanding (3) to $x_{t+1} = x_t - \eta_t(\mu m_{t-1} + g_t)$, we can see that we get NAG by replacing $m_{t-1}$ with $m_t$. Thus, Nesterov's momentum differs from the classic momentum in that it updates the model parameter using the future momentum vector. One can interpret Nesterov's momentum as an attempt to add a correction direction to the classic momentum method. NAG is argued to be more effective in the early optimization, and is more tolerant of large values of $\mu$ compared to the classic momentum method [26]. Recently, Adam with Nesterov's momentum is proposed in [6] and it shows better convergence performance than Adam on some tasks. Inspired by this change, two variants of LAMB using Nesterov's momentum are proposed in [30]. However, their modifications do not take the normalization factor into account, and the resultant algorithms do not show any improvement over LAMB. In this paper, we propose a different way to modify the momentum component of LAMB to take advantage of the superior performance of Nesterov's acceleration.

## 3 Proposed Methods

### 3.1 Normalized Gradient

In LAMB, the update is normalized by its $\ell_2$ norm. In addition to that, we propose to normalize the gradient in each block:

$$\tilde{g}_{t,\mathcal{G}_b} = g_{t,\mathcal{G}_b} / \|g_{t,\mathcal{G}_b}\|_2. \tag{4}$$

**Algorithm 2** LANS

**Input:** stepsize sequence $\eta_t$; $0 < \beta_1, \beta_2 < 1$; scaling function $\phi$; $\epsilon > 0$; regularization parameter $\lambda$.
**Initialize:** $x_1 \in \mathbb{R}^d$; $m_0, v_0 = 0$.
**for** $t = 1, \ldots, T$ **do**
  Compute mini-batch stochastic gradient $g_t$
  **for** $b = 1, 2, \ldots, B$ **do**
    $\tilde{g}_{t,\mathcal{G}_b} = g_{t,\mathcal{G}_b}/\|g_{t,\mathcal{G}_b}\|_2$
    $m_{t,\mathcal{G}_b} = \beta_1 m_{t-1,\mathcal{G}_b} + (1-\beta_1)\tilde{g}_{t,\mathcal{G}_b}$
    $v_{t,\mathcal{G}_b} = \beta_2 v_{t-1,\mathcal{G}_b} + (1-\beta_2)\tilde{g}^2_{t,\mathcal{G}_b}$
    $\tilde{m}_{t,\mathcal{G}_b} = m_{t,\mathcal{G}_b}/(1-\beta_1^t)$
    $\tilde{v}_{t,\mathcal{G}_b} = v_{t,\mathcal{G}_b}/(1-\beta_2^t)$
    compute ratios $r_{t,\mathcal{G}_b} = \frac{\tilde{m}_{t,\mathcal{G}_b}}{\sqrt{\tilde{v}_{t,\mathcal{G}_b}}+\epsilon}$ and $c_{t,\mathcal{G}_b} = \frac{\tilde{g}_{t,\mathcal{G}_b}}{\sqrt{\tilde{v}_{t,\mathcal{G}_b}}+\epsilon}$
    $d_{t,\mathcal{G}_b} = \phi(\|x_{t,\mathcal{G}_b}\|_2)\left[\frac{\beta_1}{\|r_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}\|}(r_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}) + \frac{1-\beta_1}{\|c_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}\|}(c_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b})\right]$
    $x_{t+1,\mathcal{G}_b} = x_{t,\mathcal{G}_b} - \eta_t d_{t,\mathcal{G}_b}$
  **end for**
**end for**

Then, we use $\tilde{g}_{t,\mathcal{G}_b}$ to update first-order and second-order momentums $m_t$ and $v_t$, respectively. This technique was first introduced in [31] for accelerating Adam in training deep neural networks. Using the per-block gradient normalization, the gradient clipping is no longer necessary. Ignoring the gradient magnitude makes the gradient descent methods more robust to vanishing and exploding gradients.

### 3.2 Incorporate Nesterov's Momentum into LAMB

In order to incorporate Nesterov's momentum, we first rewrite the step 11 in Algorithm 1 as

$$x_{t+1,\mathcal{G}_b} = x_{t,\mathcal{G}_b} - \eta_t \phi(\|x_{t,\mathcal{G}_b}\|_2)\left[\frac{\beta_1}{\|r_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}\|}\left(\frac{m_{t-1,\mathcal{G}_b}/(1-\beta_1^t)}{\sqrt{\tilde{v}_{t,\mathcal{G}_b}}+\epsilon}+\lambda x_{t,\mathcal{G}_b}\right)\right. \quad (5)$$

$$\left. + \frac{1-\beta_1}{\|r_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}\|}\left(\frac{g_{t,\mathcal{G}_b}/(1-\beta_1^t)}{\sqrt{\tilde{v}_{t,\mathcal{G}_b}}+\epsilon}+\lambda x_{t,\mathcal{G}_b}\right)\right]. \quad (6)$$

To apply the same trick as in Nesterov's momentum, first we substitute $m_{t,\mathcal{G}_b}$ for $m_{t-1,\mathcal{G}_b}$ in (5), and then we modify the normalization factors to ensure unit $\ell_2$-norm for both (5) and (6), leading to the following new update rule:

$$x_{t+1,\mathcal{G}_b} = x_{t,\mathcal{G}_b} - \eta_t \phi(\|x_{t,\mathcal{G}_b}\|_2)\left[\frac{\beta_1}{\|r_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}\|}(r_t+\lambda x_{t,\mathcal{G}_b})\right.$$
$$\left. + \frac{1-\beta_1}{\|a_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b}\|}(a_{t,\mathcal{G}_b}+\lambda x_{t,\mathcal{G}_b})\right], \quad (7)$$

where $a_{t,\mathcal{G}_b} = \frac{g_{t,\mathcal{G}_b}}{\sqrt{\tilde{v}_{t,\mathcal{G}_b}}+\epsilon}$. Note that we remove the factor $1/(1-\beta_1^t)$ in (6) for (7), as this factor leads to a bias towards $g_{t,\mathcal{G}_b}$ when the normalization is modified and regularization parameter $\lambda > 0$. Interestingly, the resultant update is simply a convex combination between LAMB updates with and without first-order momentum. Combining (7) with (4), we obtain Algorithm 2.

### 3.3 Learning Rate Scheduler for Large Mini-Batch

For large mini-batch optimization, warmup is usually used at the start of the training [9]. Goyal *et al.* [9] proposed to use a linear warmup in the beginning and return to the original learning rate schedule

afterwards. For BERT pretraining, LAMB uses a learning rate schedule of form [30]

$$\eta_t = \begin{cases} \eta \dfrac{t}{T_{warmup}}, & \text{if } t \leq T_{warmup} \\ \eta \dfrac{T-t}{T-T_{warmup}}, & \text{otherwise,} \end{cases} \tag{8}$$

where $\eta > 0$ is the maximum learning rate that the optimization algorithms use throughout the training and $T_{warmup}$ denotes the number of iterations in $warmup$ stage. It can be seen that $\eta_t$ gradually increases to $\eta$ when $t$ approaches $T_{warmup}$ and decreases to 0 when $t \to T$. In [30], a square root scheduling rule is proposed for increasing mini-batch size: $\eta = \sqrt{k}\tilde{\eta}$, where $k$ is the mini-batch size and $\tilde{\eta}$ is a reference learning rate for a small mini-batch size. To achieve speedup using $\tau$ times larger mini-batch size, the number of training iterations $T$ is reduced by $\tau$ times while the learning rate $\eta$ is increased by $\sqrt{\tau}$ times. Using such scheduler, LAMB successfully scaled BERT pretraining up to a mini-batch size of 32K without any accuracy loss. For a larger mini-batch size, this square root scheduling breaks as it exceeds a maximum rate that does not depend on the mini-batch size. Thus, a smaller learning rate is used for a mini-batch size of 64K with a small degradation of accuracy.

Considering that the learning rate is theoretically upper bounded by the inverse of the Lipschitz constant $L$ [17, 8, 11, 30] up to some small constants (e.g., 1 or 2), one cannot scale the learning rate indefinitely. We propose to add a $constant$ transient phase after the $warmup$ stage as shown below

$$\eta_t = \begin{cases} \eta \dfrac{t}{T_{warmup}}, & \text{if } t \leq T_{warmup} \\ \eta, & \text{if } T_{warmup} < t \leq T_{warmup} + T_{const} \\ \eta \dfrac{T-t}{T-T_{warmup}-T_{const}}, & \text{otherwise,} \end{cases} \tag{9}$$

where $T_{const}$ is the number of iterations in which a constant learning rate is used. This scheme allows the training to have sufficient progress even one cannot further increase $\eta$. Figure 1 shows

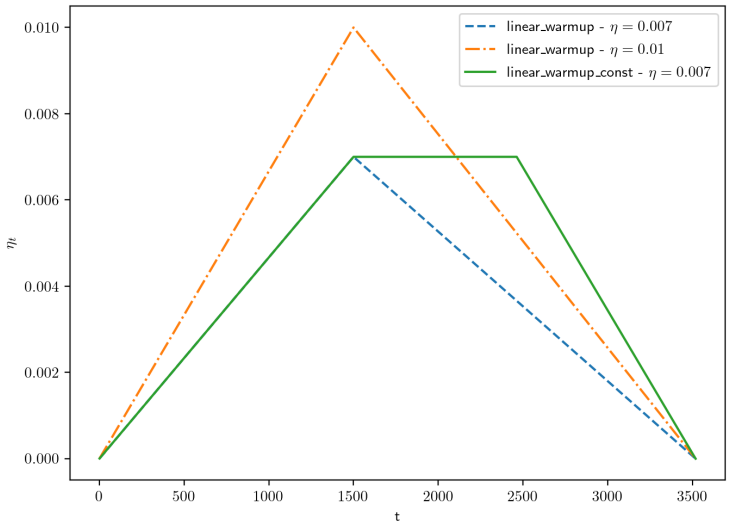


Figure 1: Visual illustrations of (8) with $\eta = 0.007, 0.01$ and (9) with $\eta = 0.007$. $T = 3519$, $T_{warmup} = 1500$, and $T_{const} = 963$.

visualization of (8) with $\eta = 0.007, 0.01$ and (9) with $\eta = 0.007$. $\eta = 0.01$ refers to the ideal learning rate that we scaled the mini-batch size from 32K to 128K. However, 0.01 has exceeded the

maximum learning rate and results in divergence. Therefore, we have to use a smaller one such as $0.007$. Nonetheless, this smaller learning rate downgrades performance. In particular, the difference between areas under curve of (8) with $\eta = 0.007, 0.01$ is $5.28$. Using the proposed schedule (9), we can reduce the difference to $1.91$.

### 3.4 Data Sharding in Distributed Training

In large-scale mini-batch training, the quality of the mini-batch plays an important role. In order to use large learning rate, one need to have as small gradient variance as possible. For example, random sampling with replacement results in a variance bound of $O(\frac{\sigma^2}{k})$ [4] while random sampling without replacement gives a better bound of $O(\frac{n-k}{k(n-1)}\sigma^2)$ [15], where $\sigma^2$ is the upper bound of the gradient variance. It can be seen that the variance only goes to zero when $k \to \infty$ for random sampling with replacement while the variance is zero when $k = n$ for random sampling without replacement. Thus, random sampling without replacement results in better efficiency of using the same mini-batch size. In distributed training, to make sure that the mini-batch does not have redundant samples, we only grant each worker access to a shard of the dataset. Within each shard, random shuffling is used to construct the mini-batch samples.

## 4 Experiments

In the experiment, we train a BERT-Large model on Wikipedia and BooksCorpus datasets. The experiment is conducted on 192 Amazon EC2 P3dn.24xlarge instances. There are $1536$ NVIDIA V100 GPUs in total. The preprocessed dataset is partitioned into $1536$ shards. The elastic fabric adapter (EFA) is enabled to improve the communication efficiency. We use LANS with the proposed learning rate schedule (9). The training is divided into 2 stages: the first $3519$ iterations is trained with a short sequence length of $128$ and the last $782$ steps is trained with a longer sequence length of $512$. We use a mini-batch size of 96K and 33K for phases 1 and 2, respectively.

Let $ratio_{warmup} = T_{warmup}/T_{stage_i} * 100\%$ and $ratio_{const} = T_{const}/T_{stage_i} * 100\%$ for $i$-th training stage. We use $ratio_{warmup} = 1.5 * ratio_{warmup_{64K}}$, where $ratio_{warmup_{64K}}$ is the warmup ratio used for LAMB with mini-batch sizes 64K/32K, and we select $ratio_{const}$ such that $ratio_{warmup} + ratio_{const} = 70\%$ and $ratio_{warmup} + ratio_{const} = 30\%$ for stages 1 and 2 training, respectively. The hyper-parameters used in the experiments are shown in Table 1. We can use larger

| | $\eta$ | $ratio_{warmup}$ | $ratio_{const}$ |
|---|---|---|---|
| stage 1 | 0.00675 | 42.65% | 27.35% |
| stage 2 | 0.005 | 19.2% | 10.8% |

Table 1: Hyper-parameters used in LANS with mini-batch sizes 96K/33K.

learning rates such as 0.00725 in the first stage, but we observed that $\eta = 0.00675$ gives better performance. For finetuning, we use AdamW optimizer [16] with per-block gradient normalization (4). The experiment result in shown in Table 2. As can be seen, the proposed methods only need $4301$

| | batch size | steps | F1 score on dev set | TPUs/GPUs | time |
|---|---|---|---|---|---|
| LAMB [30] | 64K/32K | 8599 | 90.58 | 1024 TPUs | 76.2m |
| LAMB [30] | 96K/33K | 4301 | diverge | 1536 GPUs | N/A |
| LANS | 96K/33K | 4301 | 90.60 | 1536 GPUs | 53.6m |

Table 2: Experiment results on BERT pretraining. The F1 score on SQuAD-v1.1 development set is used as the evaluation metric. The result of LANS is compared to the one of LAMB from Table 1 in [30].

iterations and finish the BERT pretraining in 53.6 minutes, while LAMB fails to further scale up the mini-batch size in BERT training. On the other hand, when the model is trained with $4301$ steps, the square root scheduling rule suggests larger mini-batch sizes of 128K and 64K for stages 1 and 2, respectively. With the proposed methods, we are able to achieve the target accuracy using much smaller mini-batch sizes. This further reduces the total computational workload and training time.

## 5 Conclusion

In this paper, we propose an accelerated large batch method called LANS. LANS employs block-wise gradient normalization and Nesterov's momentum. By identifying the insufficiency the of linear warmup learning rate schedule for large mini-batch training, we introduce a new learning rate scheduler that adopts a constant learning rate for few epochs after the warmup phase. The empirical evaluation shows that the proposed methods scale the BERT pretraining to mini-batch sizes of 96K and 33K for first and second training stages, respectively. And, they only use 54 minutes to complete the BERT training on 192 Amazon EC2 P3dn.24xlarge instances.

## References


[1] L. B. Almeida, T. Langlois, J. D. Amaral, and A. Plakhov. Parameter adaptation in stochastic optimization. In *On-line learning in neural networks*, pages 111–134. Cambridge University Press, 1999.

[2] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[3] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, and A. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[4] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1):165–202, 2012.

[5] J. Devlin, Ming-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186, 2019.

[6] T. Dozat. Incorporating nesterov momentum into adam. In *Workshop of the International Conference on Learning Representations*, 2016.

[7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7):2121–2159, 2011.

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

[9] 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.

[10] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[11] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the International Conference on Machine Learning*, pages 1225–1234, 2016.

[12] E. Hazan, K. Levy, and S. Shalev-Shwartz. Beyond convexity: Stochastic quasi-convex optimization. In *Advances in Neural Information Processing Systems*, pages 1594–1602, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[14] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations*, 2015.

[15] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670, 2014.

[16] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[17] Y. Nesterov. *Introductory Lectures on Convex Optimization*. Springer, 2004.

[18] 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.

[19] Y. E. Nesterov. Minimization methods for nonsmooth convex and quasiconvex functions. *Matekon*, 29:519–531, 1984.

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

[21] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *Proceedings of the International Conference for Learning Representations*, 2018.

[22] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[23] 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. *Journal of Machine Learning Research*, 20:1–49, 2019.

[24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[25] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor. Layer-specific adaptive learning rates for deep networks. In *Proceedings of the International Conference on Machine Learning and Applications*, pages 364–368, 2015.

[26] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning*, pages 1139–1147, 2013.

[27] T. Tieleman and G. Hinton. Lecture 6.5 - RMSProp, COURSERA: Neural networks for machine learning, 2012.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[29] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1707.03888*, 2017.

[30] Y. You, J. Li, J. Hseu, X. Song, J. Demmel, and C. Hsieh. Reducing bert pre-training time from 3 days to 76 minutes. In *Proceedings of the International Conference on Learning Representations*, 2020.

[31] A. W. Yu, L. Huang, Q. Lin, R. Salakhutdinov, and J. Carbonell. Block-normalized gradient method: An empirical study for training deep neural network. *arXiv preprint arXiv:1707.04822*, 2017.

[32] A. W. Yu, Q. Lin, R. Salakhutdinov, and J. Carbonell. Normalized gradient with adaptive stepsize method for deep neural network training. *arXiv preprint arXiv:1707.04822*, 2017.

[33] M. D. Zeiler. ADADELTA: An adaptive learning rate method. Preprint arXiv:1212.5701, 2012.

[34] Z. Zhou, Q. Zhang, G. Lu, H. Wang, W. Zhang, and Y. Yu. Adashift: Decorrelation and convergence of adaptive learning rate methods. *arXiv preprint arXiv:1810.00143*, 2018.

[35] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Neural Information Processing Systems*, pages 2595–2603, 2010.