How to achieve large batch size training of a DNN with SGD by adjusting hyperparameters?

Loïc Signer, Clément Barbier
CS-439 Optimization for Machine Learning, EPFL, Switzerland

***Abstract*—A very common way to enable the training of deep neural networks on very large datasets is to use the parallelization capabilities of GPUs. For this, it may be necessary to increase the size of the minibatches used in Stochastic Gradient Descent (SGD) up to 8192. However, it is difficult to keep the accuracy and convergence speed when the minibatches become too large. Our method consists in analyzing various methods of selection of hyper-parameters used in SGD (learning rate, momentum) according to the batch size in order to increase the batch size without degrading the performance of the DNN. To achieve our results we will use the LeNet5 Convolutional Neural Network (CNN) with two convolutional layers and conduct the experiments on the MNIST dataset.**

I. Introduction

We start by reviewing the principles of SGD which is the heart of our following discussions. The purpose of the training of a DNN is to find the optimal parameters w to minimize the following loss function: $L(w) = \frac{1}{N}\sum_{i=1}^{N} l(x_i, w)$. Here our training dataset X composed of N samples x_i. During the training, each sample is associated with a loss $l(x_i, w)$ which is the cross-entropy loss during all our report from the samples. The SGD with minibatches performes the following update on the weights w of the DNN at the t-th iteration to minimize $L(w)$: $w_{t+1} = w_t - \eta \frac{1}{b}\sum_{x \in B_t} \nabla l(x, w_t)$.

Let us refer to the batch of samples at t-th iteration as B_t and η is the learning rate. We have $|B_t| = b$. In the following, we will use the notation $g_{i,t} = \nabla l(x_i, w_t)$. Note that in practice a momentum is also considered in the SGD, for the moment we consider the SGD without momentum and we will study its importance in the section III.

This simple algorithm to train DNN has led to improved results in a wide variety of applications such as computer vision and NLP. However one major limitation of DNN is that the training can be an extremely time-consuming process especially with very large datasets having sizes much larger than the Gigabyte. To scale up the DNN training to large datasets, the natural strategy is to use more computational power with several GPUs enabling mini-batch SGD training with data-parallelism. But the use of multiple GPUs leads to the increase of the batch size to keep the per-GPU work constant. However as pointed-out by [1], training models with large batch size diminishes the accuracy and even when the models were trained without limits (i.e until the loss function ceased to improve). In our report, we will experiment methods to reduce this accuracy error when the batch size becomes too large via the value of the learning rate and the momentum. For that we will use the CNN LeNet5 [2], we will use only one GPU for the training of the model since the objective here is simply to reduce the negative effect of large batch sizes. Finally we will use the MNIST dataset which consists of a training set of 60K and a test set of 10K 28*28 gray-scale images representing digits ranging from 0 to 9.

II. Compensation of large batch sizes with learning rates

First, before studying how to tune the learning rate according to the minibatch size, we wanted to select a default learning rate value $\eta_0 = 0.01$. This value was obtained by selecting the learning rate giving the best test accuracy after 30 epochs with a batch size of $b_0 = 64$ on the MNIST data set. On the figure 1, it is easy to see the degradation of the performance of the model with the increase of the batch size, this degradation is present in 2 ways on the optimization of the loss function, first the convergence is slower and moreover the convergence is done towards a less good minimum.

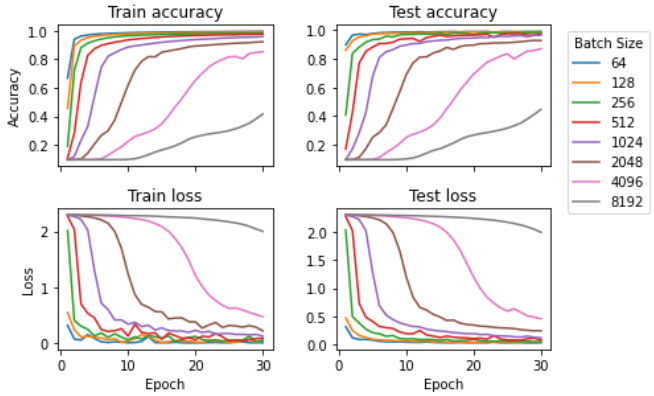

Fig. 1: LeNet5 with learning rate $\eta_0 = 0.01$ when increasing batch sizes

Our purpose is to use larger minibatches while keeping training and testing accuracy without needing more epochs.

The most studied strategy in the literature is to adjust the learning rate to the batch size. The first strategy called the **sqrt scaling rule** analysed by [3] is to keep the variance of the weights updates constant by maintaining constant the covariance matrix of the parameters update step $\Delta w_t = w_{t+1} - w_t$ for all mini-batch sizes. With some calculations, assuming uniform sampling of the mini-batch indices, [3] shows:

$$cov(\Delta w_t, \Delta w_t) = \eta^2 (\frac{1}{M} - \frac{1}{N}) \frac{1}{N} \sum_{i=1}^{N} g_{i,t} g_{i,t}^{\top}$$

Which implies when $M \ll N$:

$$cov(\Delta w_t, \Delta w_t) \approx \frac{\eta^2}{M}(\frac{1}{N}\sum_{i=1}^{N} g_{i,t} g_{i,t}^{\top})$$

Thus to keep the variance of the weights updates constant for a batch size b, we need to use $\eta = \eta_0 \sqrt{\frac{b}{b_0}}$.

The second possible strategy developped by [4] and [5] is to set up a **linear scaling rule**, $\eta = \eta_0 \frac{b}{b_0}$. The intuitive justification of this scaling rule is to assume that k successive SGD iterations with small minibatches of size b_0 and learning rate η_0 is close that a single SGD iteration with the union of these minibatches and a k times bigger learning. Indeed during our k iterations we have:

$$w_{t+k} = w_t - \eta_0 \frac{1}{b_0} \sum_{j<k} \sum_{x \in B_j} \nabla l(x, w_{t+j})$$

On the other side using 1 SGD iteration with a learning rate η and a minibatch $B = \cup_{j=1}^{k} B_j$ so $b = k b_0$, we have:

$$w_{t+1}^{'} = w_t - \eta \frac{1}{b} \sum_{x \in B} \nabla l(x, w_t)$$

But under the approximation $\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$, with $\eta = k\eta_0$, we get $w_{t+1}^{'} \approx w_{t+k}$. This assumption does not seem accurate in the general case but empirical results shown by [5] this approximation does not degrade the training of DNN using larger minibatches and the linear scaling rule.

However as pointed out by [4], the previous approximation $\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$ clearly does not hold during the first iterations of the training where the weights updates are more important. To solve this problem in the early stage of training, two warmup strategies are proposed. The **constant warmup** strategy consists in using a low learning rate $\eta = \eta_0$ for the first epochs of training and then returning to the linear scaling rule $\eta = \eta_0 \frac{b}{b_0}$. The second possibility is to use a **gradual warmup** during the first k epochs to gradually increase the value of the learning from η_0 to the linear scaling rule: $\eta = \frac{k+1-i}{k}\eta_0 + \frac{i-1}{k}\frac{b}{b_0}\eta_0$ for $1 \leq i \leq k$. The gradual warmup aims at avoiding a sudden massive increase of the learning rate.

III. Impact of Momentum on batch sizes

In recent years, DNN optimization algortims have greatly improved, notably the SGD now takes into account momentum to improve its performance [6]. The very popular momentum SGD algorithm works as follows with m the momentum parameter ($m \in [0, 1[$):

$$\begin{cases} v_{t+1} = m v_t + \frac{1}{b} \sum_{x \in B_t} \nabla l(x, w_t) \\ w_{t+1} = w_t - \eta v_{t+1} \end{cases}$$

[7] explains that the adding of the momentum m leads to an effective learning rate $\eta_{eff} = \frac{\eta}{1-m}$, the intuitive explanation behind this is the fact that each minibatch is taken in the following iterations by a factor that is multiplied each time by m. Since the limit of the sum of the geometric sequence with common ratio m is $\frac{1}{1-m}$ when $m \in [0, 1[$, we retrieve the multiplication by $\frac{1}{1-m}$ in η_{eff}. As a result, when taking into account the momentum, with the linear scling rule, the new effective learning rate should be $\eta_{eff} = \eta_{eff_0} \frac{b}{b_0}$ with η_{eff_0} the reference effective learning rate for a small batch size b_0.

IV. Results

To study the impact of the increase of the learning with the batch size, we started again from the results of the baseline in figure 1. Here again we used as reference size $b_0 = 64$ and a reference learning rate $\eta_0 = 0.01$. Then we tested with a training on 30 epochs of the CNN LeNet5 [2] on the MNIST Dataset for minibatch sizes corresponding to all powers of 2 between 64 and 8192. Figures 2 and 3 show respectively the impact of sqrt and linear scaling rules on the evolution of the accuracy during the training of the LeNet5 model.

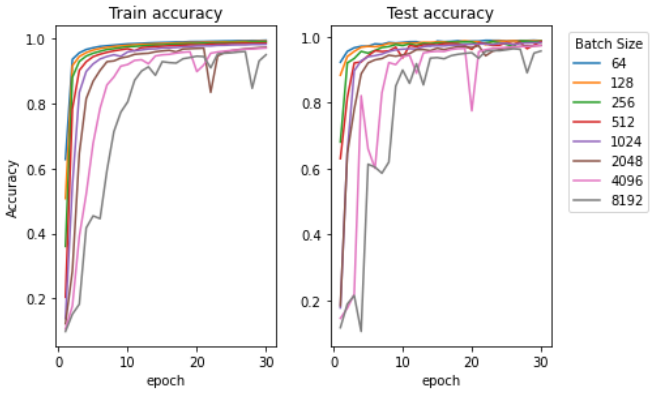

Fig. 2: Accuracy of LeNet5 with sqrt scaling rule with $b_0 = 64$ and $\eta_0 = 0.01$

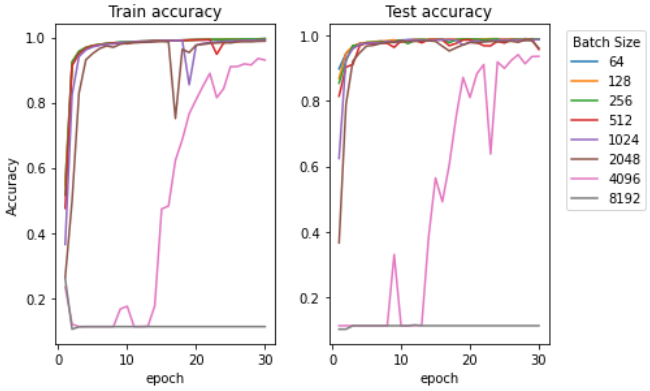

Fig. 3: Accuracy of LeNet5 with linear scaling rule with $b_0 = 64$ and $\eta_0 = 0.01$

The sqrt and linear scaling rule show very well that the increase of the learning rate with the batch size increases very quickly the speed of convergence for the largest batch sizes. The article mentioned that despite the theoretical justifications to use the sqrt scaling rule, the linear scaling rule was more efficient in practice. In our case, we find only half of these discoveries, indeed the linear scaling rule is more efficient for batch sizes up to 1024 but beyond that the learning rate becomes much too important and prevents any convergence. This is the reason for the better results of the sqrt learning rule for the largest sizes (4096 and 8192)

Then, in order to improve the convergence of large batch sizes (especially 4096 and 8192) with the linear scaling rule, we implemented gradual warmup strategy 4 in order to limit the difference between the model weights between 2 successive iterations by limiting the learning rate for the

first iterations hoping that the too large learning rate will not make our model diverge anymore if the gradients of the loss function are small enough near the minimum. We have chosen to use the warmup strategy for the first 10 epochs so that the increase is not too abrupt and thus allow the model to train correctly on the first epochs for the largest batch sizes.

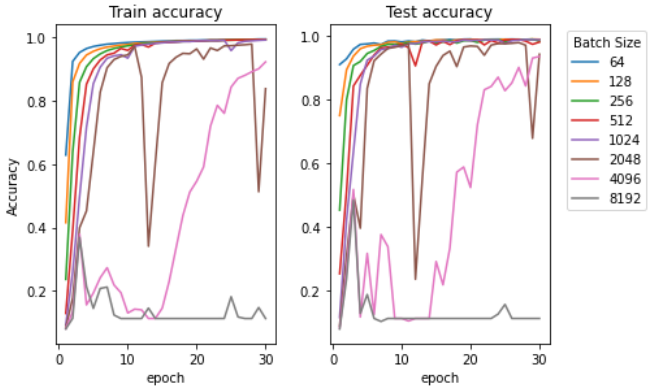

Fig. 4: Accuracy of LeNet5 with gradual warmup on 10 epochs and linear scaling rule with $b_0 = 64$ and $\eta_0 = 0.01$

It is possible to notice in 4 that if initially while the learning rate is increasing linearly the convergence is efficient for the big learning rates, as soon as the value of this one becomes too important, the model starts to diverge even if the previous iterations had been very efficient. That's why here the warmup strategy doesn't seem to give any significant improvement.

In addition, we studied how the use of momentum could be used to enable better convergence for large batch sizes. For this, we were inspired by the heursitic "How to achieve large batch training" method presented by [8]. We set the learning rate to $\eta_0 = 0.01$ and the momentum coefficient to $m_0 = 0.9$, and trained the LeNet5 model with those parameters (see figure 5). Then we started from the biggest batch size that get the optimal accuracy and quick convergence on the test accuracy (2048 in 5), then we repeatedly increase the batch size by a factor of 2, while scaling the learning rate η with the sqrt scaling rule until until the validation set accuracy started to fall. Finally we repeatedly increased the batch size by a factor of 2, while scaling the momentum such that $\frac{1}{1-m}$ follows the sqrt scaling rule, indeed we want $\eta_{eff} = \frac{\eta}{1-m}$ follows the desired scaling rule.

The first remark to make is that momentum significantly improves the convergence of our model whatever the size of the minibatches (see 5). By increasing η, we see in 6 that the improvement as expected for $b = 4906$ but that for $b = 8192$ the convergence becomes unstable because of a too important learning rate. Moreover, we finally notice that the addition of sqrt scaling of the momentum on $b = 8192$ allows a faster convergence but seems to be very unstable as shown by the decrease of the accuracy at epoch 8. Thu s we have just shown that the method proposed by [8] to adjust the momentum and the learning rate, thus allows to obtain even faster convergences but which seem to be unstable.

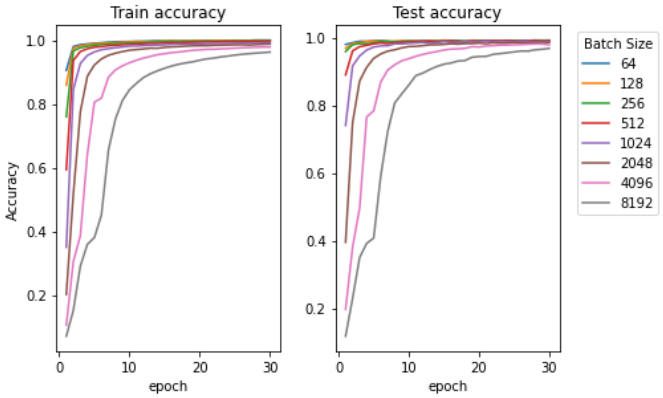

Fig. 5: Accuracy of LeNet5 with $m_0 = 0.9$ and $\eta_0 = 0.01$

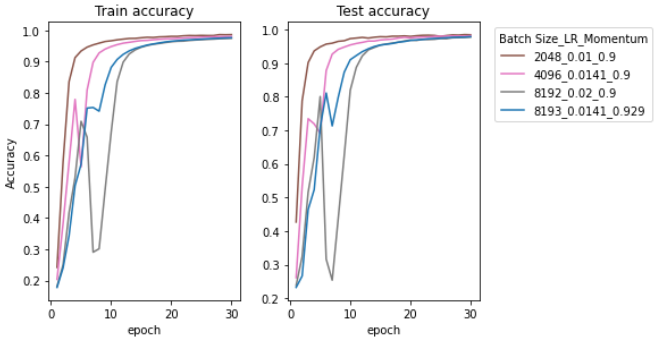

Fig. 6: Accuracy of LeNet5 with sqrt scaling strategy with learning rate then momentum starting with $b_0 = 2048$, $m_0 = 0.9$ and $\eta_0 = 0.01$

V. Discussion

In our model we have primarily discussed the improvement of convergence of DNN models when large batch sizes are used during SGD with the scaling rules on learning rate and momentum. It would be necessary to conduct a similar analysis on the Adam optimizer and check if the proposed scaling rule is still valid with the hyperparameters of this optimization method. Moreover we have noticed a strong instability when using too high learning rates, a method not tested here would be to use a decreasing learning rate with each epoch to reduce these convergence problems. Finally, it would be necessary to measure the impact on the training time of using multiple GPUs using this scaling method in order to better study the computation time/model accuracy trade-off. For this, it would be necessary to carry out the same study but this time on a dataset much larger than MNIST such as ImageNet for example where the use of large batch sizes would be more relevant for training the model.

VI. Summary

Thus, during our DNN training, we could see that increasing the learning rate as well as the momentum according to the sqrt or linear scaling rule could significantly improve the convergence for large batch sizes up to 8192. This increase can significantly reduce DNN training times if multiple GPUs are used to use data-parallelism during stochastic gradient descent.

REFERENCES

[1] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2016.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[3] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," 2018.

[4] 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," 2018.

[5] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017.

[6] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: http://proceedings.mlr.press/v28/sutskever13.html

[7] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," 2017.

[8] S. L. Smith and Q. V. Le, "A bayesian perspective on generalization and stochastic gradient descent," 2017.