Efficient Dynamic Batch Adaptation (Student Abstract)

Cristian Simionescu, George Stoica

”Alexandru Ioan Cuza” University
cristian@nexusmedia.ro, sgeorge.sstoica99@gmail.com

Abstract

In this paper we introduce Efficient Dynamic Batch Adaptation (EDBA), which improves on a previous method that works by adjusting the composition and the size of the current batch. Our improvements allow for Dynamic Batch Adaptation to feasibly scale up for bigger models and datasets, drastically improving model convergence and generalization. We show how the method is still able to perform especially well in data-scarce scenarios, managing to obtain a test accuracy on 100 samples of CIFAR-10 of 90.68%, while the baseline only reaches 23.79%. On the full CIFAR-10 dataset, EDBA reaches convergence in $\sim$120 epochs while the baseline requires $\sim$300 epochs.

Introduction

As current hardware has evolved allowing the use of bigger batch sizes in order to speed up the training, several researchers studied how to enable the usage of bigger batches in efficient ways. Using the inverse relation between learning rate and batch size, some studies went in the direction of increasing the batch size instead of using other methods of decreasing the learning rate (Devarakonda, Naumov, and Garland 2017) (Khan et al. 2020) (Liu et al. 2019) (Simionescu, Stoica, and Herscovici 2022).

When we have access to bigger datasets deep learning models become more powerful. However, in data scarce environments deep learning methods are not able to perform as well, even with the usage of transfer learning. Moreover, several niche domains do not find transfer learning suitable for their usage. In this regards, the work done by (Simionescu, Stoica, and Herscovici 2022) goes in this direction since their proposed algorithm Dynamic Batch Adaptation (DBA), is able to learn and generalize well provided few samples from a dataset.

Related Work

DBA, introduced in (Simionescu, Stoica, and Herscovici 2022), is a method of dynamically choosing the composition of a batch size, by selecting a subset of samples to be used for updating the weights of the network according to a certain criteria. This was done by maintaining per-sample gradients during backpropagation of the current batch. Consequently, the per-sample gradients were used to minimize a *margin-based loss* which consists of a *model metric*, an indicator of the current fitness of the model, and a *gradient metric*, which measures how suitable for updating is a subset of per-sample gradients from the current batch. They propose a selection that searches for a subset of the gradients that minimizes the absolute value of the *margin-based loss*, which is later used in the update step. Moreover, the subset of gradients is calculated separately for each layer of the model.

The method was validated on the MNIST dataset which shows DBA to greatly outperform baseline methods in the context of a extremely data scarce environment (using small subsets of MNIST), managing to achieve 97.79% test accuracy training on only 1% of the MNIST dataset, with no data augmentation, representing a relative error rate reduction of 81.78% compared to the standard optimizers, SGD (Stochastic Gradient Descent) and Adam.

However, Dynamic Batch Adaptation method has multiple drawbacks. By using per-sample gradients and calculating the backpropagation with regards to each sample from the batch, it increases the number of parameters of the model by a factor of batch size, and the computation is slower due to the lack of parallelization of the gradient selection process. This makes the method unusable for deeper models in terms of both memory and compute time.

In this paper, we address these issues, greatly improving the performance of DBA, designing a better selection process, adjusting the metrics used to select the gradients and scaling the method on bigger models and larger datasets, CIFAR10 and CIFAR100. We named the new method EDBA (Efficient Dynamic Batch Adaptation).

Improvements

We have decided to remove the selection of gradients used for update on each layer in order to reduce the additional parameters needed and the computation time required for computing per-sample gradients. In EDBA, we replace the selection of per-sample gradients on each layer by doing an initial forward step though the model, we use this to calculate per-sample gradients for only the last layer. As a result we are able to apply a selection method on those per-sample gradients, those samples are thereafter taken and used in an

		Test Accuracy %			
Dataset	Method	100% of Data	10% of Data	1% of Data	100 Samples
CIFAR10	Baseline	92.014 ± 0.115	71.654 ± 0.087	40.516 ± 1.358	23.796 ± 1.092
CIFAR10	EDBA	**92.196 ± 0.219**	**91.648 ± 0.171**	**91.514 ± 0.224**	**90.688 ± 0.168**
CIFAR100	Baseline	67.308 ± 0.197	29.628 ± 0.862	8.404 ± 0.403	—
CIFAR100	EDBA	**70.550 ± 0.097**	**69.138 ± 0.264**	**68.852 ± 0.341**	—

Table 1: Results using $x\%$ of the data

usual forward and backpropagation step. By doing this, we reduced the cost of calculating per-sample gradients and the selection process for each layer in DBA to a forward step for the entire batch and a single selection process applied on the last layer in EDBA.

We have replaced the selection process used in DBA, a sequential removal or addition of samples, with a new procedure. We choose which samples from the current batch to use by randomly generating subsets of the entire batch using an uniform distribution, and selecting the subset which minimizes a *margin-based loss*. We use the same *margin-based loss* as DBA, the slope between the *model metric* and the *gradient metric* using the exponential running average of the two values as seen in 1.

$$\left| \frac{ModelMetric - ModelMetric_{exp}}{GradMetric - GradMetric_{exp}} - 1 \right| \quad (1)$$

The *model metric* is the loss of the model calculated for the current batch, while the *gradient metric* is the norm of the per-sample gradients on the last layer. Since the *gradient metric* is the only one that can change, our selection process chooses a subset which has the *gradient metric* following the evolution of the model loss the closest. Intuitively, our selection method chooses a subset of samples which would generate an appropriate update, meaning that the gradient metric will be selected in accordance of how well fitted the model is. At the beginning of training when the loss is high, it will allow for higher gradient magnitudes and will tend to prefer smaller batches, while toward the end of training, when the loss is small, it will be forced to raise the batch size and filter out noisy samples in order to keep the step magnitude small. That is due to the fact that when the samples in a subset have different directions, the norm of the mean of the gradients is closer to zero.

Similarly to DBA, we increase and decrease the batch size as needed. However we look at *margin-based loss* statistics directly to make the decision not at the historical number of gradients chosen.

Experiments

We have done our experiments using PreResNet56 on CIFAR10 and CIFAR100. We have used SGD as an optimizer, with a learning rate of 0.001, Nesterov momenntum of 0.9 and weight decay of 0.00001. We used an initial batch size of 100, with EDBA being able to modify it as needed. We have only used random crop and horizontal flip data augmentations, and our experiments were done on 1%, 10% and the whole dataset for both datasets. In addition, for CIFAR10 we also trained using 100 samples. All our experiments were repeated using 5 random seeds. The baseline we compared with, used the same configuration, with the exclusion of the EDBA algorithm.

We have observed a great improvement in accuracy compared to the baseline the less samples we use, as seen in Table 1. We empirically show that our method of batch filtering helps the model to maintain a good direction and generalize very well with few samples. The batch size is capped at both ends by 10 and 1000, the later being adjustable depending on the hardware used for training. Additionally, EDBA will converge considerably faster when using the full datasets, reaching convergence in $\sim$80-120 epochs while the baseline requires $\sim$300-350 epochs.

Conclusion and Future Work

We have improved DBA and have achieved high accuracy in data scarce scenarios for bigger datasets. Our results show that batch filtering brings enormous benefit in such scenarios. Furthermore, our method enabled achieving high accuracy in less epochs using dynamically adjustable batch size, decreasing the number of epochs needed until convergence.

Our next direction of research includes testing EDBA using noisy datasets and check whether our sample selection method is able to filter noisy samples and outliers when training. We also intend to test our method on non computer vision tasks and do a performance and speed analysis of our method compared with other methods used in data scarce scenarios.

References

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

Khan, W.; Ali, S.; Muhammad, U. K.; Jawad, M.; Ali, M.; and Nawaz, R. 2020. AdaDiffGrad: An Adaptive Batch Size Implementation Technique for DiffGrad Optimization Method. In *2020 14th International Conference on Innovations in Information Technology (IIT)*, 209–214. IEEE.

Liu, B.; Shen, W.; Li, P.; and Zhu, X. 2019. Accelerate mini-batch machine learning training with dynamic batch size fitting. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

Simionescu, C.; Stoica, G.; and Herscovici, R. 2022. Dynamic Batch Adaptation. *arXiv preprint arXiv:2208.00815.*