Performance Improvements of Deep Learning Networks using PyTorch

Geri Nicka
Advanced Computing Research Centre, University of Bristol.
(Dated: August 20, 2021)

1. INTRODUCTION

Training deep learning models on very large datasets is challenging and expensive, taking anywhere from hours to weeks to complete. To overcome this problem, large clusters of GPU accelerators are typically used to divide the overall task in order to reduce training time by harnessing the combined computational power of multiple accelerators working on the same problem simultaneously. Academics and researchers from various disciplines rely on machine learning to deliver novel results in their discipline and often need technical consultation from experts in the field. In this work, the PyTorch machine learning framework was used to research new methods of accelerating training time for various deep learning networks on the BlueCrystal4 and BluePebble superclusters without or minimally affecting validation accuracy. The results and discussion in this report aim to advise researchers on deep learning practices which have the potential to speed up the training time of their models and negligibly affect their accuracy.

2. PYTORCH AND OTHER MACHINE LEARNING FRAMEWORKS

The most popular machine learning languages are PyTorch, Keras and TensorFlow. There are various parameters that lead researchers to using a specific library according to their priorities. In the case of non-experts, the choice ultimately comes to technical background, requirements and ease of use. Keras is mostly used for rapid prototyping, small datasets and offers multiple back-end support. TensorFlow points towards larger datasets and is great for high performance. On the other hand, PyTorch has gained immense popularity from academics and researchers due to its simplicity compared to the other two, which require deeper technical knowledge. PyTorch also offers optimization of custom expressions which allow algorithms to run faster and allows easier debugging. The fact that PyTorch has attracted many academics led to the exploration of these custom expressions in this work.

3. GPU TRAINING

GPUs are optimized for training artificial intelligence and deep learning models as they can process multiple computations simultaneously. They possess a large number of cores, which results in better computation of multiple parallel processes. Moreover, computations in deep learning need to handle huge amounts of data, making a GPU's memory bandwidth most suitable. On the other hand, there are limitations to the usage of GPUs in deep learning, such as on package memory and data loading. A deep learning network needs to be loaded on the GPU along with a user defined batch size for training. For very large datasets the batch size cannot be optimally large as it may exceed on package memory. To overcome this limitation smaller batch sizes are set at the cost of performance. Moreover, the choice of smaller batch sizes results in the loading of more batches of data on the GPU which also increases computational time. In this work, a variety of custom expressions and parameters within the PyTorch framework were explored with the aim to increase performance during GPU training. More specifically, pinned memory, number of workers and batch size were tested to find the optimal training settings. Pinned memory is enabled by default in PyTorch and ensures faster CPU to GPU memory copy operation. To demonstrate this, five deep learning networks, namely $VGG16BN$, $ResNet18$, $AlexNet$, $SqueezeNet$ and $DenseNet$ were chosen to train the $CIFAR10$ dataset. By disabling pinned memory an average increase of 40 percent in computational time was found, justifying the default settings. It was also found that pinned memory is more effective for larger networks as it decreases the computational time of larger networks by a greater percentage. The largest network tested with disabled pinned memory was $VGG16BN$, which showed a 60 percent increase in computational time. The number of workers parameter denotes the number of processes that generate batches in parallel. No significant increase in performance was observed by increasing the number of workers. The most important parameter out of all three was found to be the batch size. It was found that there is an optimal batch size for which validation accuracy is maximum and training time is minimum. Very small or large batch sizes resulted in abnormally long training periods for one training epoch and very small accuracy. For the $CIFAR10$ dataset the optimal training batch size was determined to be 128 which resulted in an accuracy of 71 percent and training duration of 3 minutes for 14 epochs. In general, for most datasets and networks optimal training batch sizes will be 64 or 128 [1].

4. BLUECRYSTAL4 AND BLUEPEBBLE COMPARISON

The University of Bristol supercomputer network clusters BlueCrystal4 and BluePebble were also tested to determine the training times for various deep learning networks for the $CIFAR10$ and $CIFAR100$ datasets. BlueCrystal4 is equipped with the Nvidia Tesla P100 GPUs whereas BluePebble is equipped with the Nvidia RTX 3080Ti GPUs. The two datasets were to used to train 5 different networks and benchmark them. These were $VGG19$, $VGG16BN$, $ResNet18$, $SqueezeNet$ and $AlexNet$.

Model	Time (min)	Accuracy (%)	Memory (Gb)	Memory (%)
VGG16BN	5.3	72	11.3	71
SqueezeNet	5.3	75	2.2	14
AlexNet	5.5	73	1.1	7
VGG19	5.3	74	2.1	13
ResNet18	5.6	75	2.5	16

TABLE I. Training benchmarks for the CIFAR10 dataset on BlueCrystal4 for 1 training epoch with pre-trained models.

Model	Time (min)	Accuracy (%)	Memory (Gb)	Memory (%)
VGG16BN	5.6	69	10.1	84
SqueezeNet	3.7	10	2.1	18
AlexNet	3.6	66	1.0	8
VGG19	5.5	77	2.1	18
ResNet18	4.0	58	2.3	19

TABLE II. Training benchmarks for the CIFAR10 dataset on BluePebble for 1 training epoch with pre-trained models.

Model	Time (min)	Accuracy (%)	Memory (Gb)	Memory (%)
VGG16BN	5.3	53	12.2	76
SqueezeNet	5.2	53	2.4	15
AlexNet	5.7	54	1.3	8
VGG19	6.1	53	2.3	14
ResNet18	5.3	53	2.7	17

TABLE III. Training benchmarks for the CIFAR100 dataset on BlueCrystal4 for 1 training epoch with pre-trained models.

Model	Time (min)	Accuracy (%)	Memory (Gb)	Memory (%)
VGG16BN	5.5	43	10.7	90
SqueezeNet	3.6	11	2.6	22
AlexNet	3.6	42	1.9	16
VGG19	5.5	50	2.3	19
ResNet18	4.0	46	8.8	73

TABLE IV. Training benchmarks for the CIFAR100 dataset on BluePebble for 1 training epoch with pre-trained models.

The tables above demonstrate the training time for one epoch and the validation accuracy for each model. In addition, the GPU memory occupied was also monitored using the 'nvidia-smi' command and is provided in the tables as absolute and as a percentage of the overall GPU memory. It was found that BluePebble is generally more high performing that BlueCrystal4, being faster by an average of 40 percent. It is important the mention that the networks were pre-trained, helping achieve relatively high accuracies even after the first training epoch. Therefore, it is recommended that pre-trained models are implemented by researchers to train their own custom dataset. This due to the fact that pre-trained models have been already trained for weeks on very large datasets, rendering them more sophisticated than untrained models. To validate this, the untrained versions of the same networks were trained with the same datasets from scratch. Validation accuracies in this case ranged from 1% for small networks to 22% for larger more sophisticated networks after only one epoch. The point of convergence in accuracy was determined by testing a different number of epochs until accuracy did not significantly improve without increasing computation time. For the present networks and datasets convergence was achieved after 100 epochs of training for both the pre-trained and untrained versions. The average accuracies for the pre-trained and untrained versions after convergence is shown in the following table.

Dataset	BC4 Accuracy (%)	BP Accuracy (%)
CIFAR10-Trained	77.6	67.0
CIFAR10-Untrained	10.5	25.4
CIFAR100-Trained	69.2	52.4
CIFAR100-Untrained	9.6	9.0

TABLE V. Training benchmarks for the CIFAR10 and CIFAR100 datasets on BlueCrystal4 and BluePebble for pre-trained and untrained models after convergence in 100 training epochs.

It is recommended that the whole GPU is utilised by only one user when training a deep learning network so that memory is not shared with any other tasks. In this way memory can be accurately monitored and results will be consistent. As expected, it was found that large networks like $VGG16BN$ occupy more memory than smaller networks like $SqueezeNet$. It is important to mention the use of different optimizers like the ADAM and SGD optimizers. SGD generalizes better than ADAM but the latter converges faster [2]. Overall, the SGD optimizer results in a greater overall performance [3]. In this work the SGD optimizer was applied. Typical learning rates for the SGD optimizer are 0.001, 0.01 or 0.1. It is easy to determine the suitable learning rates by comparing the results they yield for one training epoch.

5. AUTOMATIC MIXED PRECISION

Mixed precision is the use of both 16-bit and 32-bit floating-point types in a model during training to accelerate it and consume less memory. By keeping certain parts of the model in the 32-bit types for numeric stability, the model has a lower step time and trains equally as well in terms of the evaluation metrics such as accuracy[4]. PyTorch offers automatic mixed precision capabilities but their implementation may be difficult for non experts depending on their model and dataset. Advanced models need to calculate many parameters and weights and automatic mixed precision requires deep expertise. The effects of mixed precision dominate for very large models and datasets where many weights need to be calculated. It is proposed that a modern and complex model like $VGG19BN$ is trained with the ImageNet dataset with and without mixed precision to compare their performance and accuracy.

[1] Mishkin, D., Sergievskiy, N., Matas, J. (2017). Systematic evaluation of convolution neural network advances on the Imagenet. Computer Vision And Image Understanding, 161, 11-19. https://doi.org/10.1016/j.cviu.2017.05.007.

[2] Nowozin, S., Sra, S., Wright, S. J. (2012). Optimization for Machine Learning. United Kingdom: MIT Press.

[3] Polyak, B., Juditsky, A. (1992). Acceleration of Stochastic Approximation by Averaging. SIAM Journal On Control And Optimization, 30(4), 838-855. https://doi.org/10.1137/0330046.

[4] Nandakumar, S., Le Gallo, M., Piveteau, C., Joshi, V., Mariani, G., Boybat, I. et al. (2020). Mixed-Precision Deep Learning Based on Computational Memory. Frontiers In Neuroscience, 14. https://doi.org/10.3389/fnins.2020.00406.