Impact of Batch Size on Performance of TensorFlow Based Convolutional Neural Networks

Duncan Campbell

Abstract—Neural networks are an emerging technology of great interest. Many neural networks are implemented using the TensorFlow library. TensorFlow abstracts the process of constructing and training neural networks, and has supports for several parallel computing interfaces including Compute Unified Device Architecture (CUDA) and Direct Machine Learning (DirectML). However, the structure and training process may not be able to take full advantage of the underlying hardware due to how TensorFlow parallelizes computations. In this paper, we examine how changing a training variable, the batch size, affects the performance of a sample Convolutional Neural Network (CNN). We use the CNN to classify images from the Modified National Institute of Standards and Technology (MNIST) database of handwritten digits. We run the CNN on a desktop with eight central processing unit (CPU) cores plus 4608 graphics processing unit (GPU) cores and on a high-performance computing (HPC) system with 2x18 CPU cores plus 5120 GPU cores. According to the experimental results, larger batch size may reduce training time with negligible or recoverable losses in accuracy for certain CNN parameters. Results also suggest that optimization must be taken on a case-by-case basis to determine the best parameters.

Keywords—Batch size, convolutional neural networks, parallel computing, TensorFlow, training time

I. Introduction

In the fields of deep learning, the optimization of neural networks stands as a critical quest for enhancing model performance across various applications. CNNs have become an area of intensive research as they have novel applications and have become more accessible in recent years due to growing access to increasingly powerful computer systems and programming libraries, such as TensorFlow, which abstract the mathematical and computational complexity of operating these networks. TensorFlow itself subdivides the tasks involved with using a neural network into simple operations which are dispatched to an underlying high-performance math library such as CUDA, DirectML, or Intel Math Kernel Library (MKL). While TensorFlow itself has many optimizations for organizing and scheduling these operations, it can still be bottlenecked by how the task is programmed and the parameters are supplied to the task. This paper examines how altering one of the training parameters, the batch size, affects the performance of the training process. The batch size affects the maximum parallelism that can be achieved during training, and is observed to impact training time, particularly on highly parallel systems such as GPUs. This research seeks to provide valuable insights that can guide practitioners and researchers in making informed decisions to harness the full potential of CNNs in diverse machine learning tasks.

II. Background

Convolutional neural networks have emerged as a core in the field of computer vision, image recognition, and object detection. TensorFlow, a machine learning framework from Google, has played a pivotal role in enabling the implementation and training of complex neural network architectures, including CNNs. TensorFlow's optimizations are described in greater detail in its 2015 whitepaper [1]; one key point is that TensorFlow builds its computation on a single fundamental unit called a tensor, which is simply a multidimensional array or matrix. TensorFlow can identify certain devices on a system which can perform accelerated computations on these tensors and will perform the appropriate work scheduling when an operation on a tensor is requested. It attempts to dispatch as many operations as possible and uses a dependency graph to identify which operations must complete before others begin. TensorFlow will dispatch these operations to any available devices it can identify [2], including CPUs, GPUs, and specialized Tensor Processor Units (TPUs). TensorFlow can use these devices to perform the operations needed for the process of training a model or to deploy one in an application.

The process of training a neural network can be roughly described in two different steps performed repeatedly [3]. First, forward propagation, where a sample of training data is fed through the network to get predicted outputs. Second, backpropagation, where the error between the predicted and actual output is fed to an optimization algorithm which will then modify the weights and bias within the network to bring the output of the model closer to the expected values.

TensorFlow employs two types of parallelism in operations, model parallelism and data parallelism [3]. Model parallelism is parallel computations involved in the process of propagating values through the network (e.g., the multiplication of vectors and matrices), which is performed by the underlying math library. Data parallelism is the processing of multiple pieces of training data in parallel. TensorFlow performs data parallelism by dividing a training dataset into batches of a certain size and will attempt to process each batch in parallel. The accumulated error from the entire batch is then used by the optimization

algorithm to update the weights. Figure 1 shows two variations of how TensorFlow may accomplish data parallelism (namely, synchronous and asynchronous). TensorFlow dispatches training data to each device by accumulating the error and synchronously updating the parameters of the model or by using multiple training clients asynchronously generating updates.

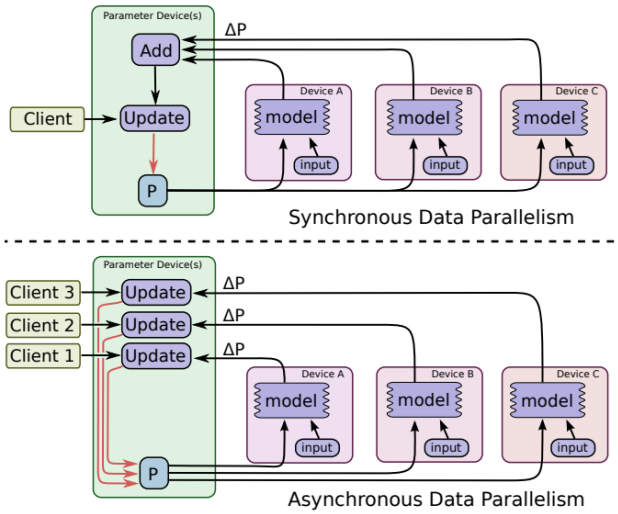

Figure 1. TensorFlow's use of data parallelism [1]

Backpropagation, however, cannot be parallelized to the same degree as forward propagation within the synchronous training process because it cannot take full advantage of data parallelism; it is only run once after each batch and must run after each batch is entirely processed. The processing of the batch, however, can theoretically be parallelized up to the size of the batch; each piece of data can be processed independently of each other and summation of the error is a common reduction operation. Therefore, for neural networks in which the computational cost of forward propagation for an individual piece of data is low, the limiting factor in training speed will be the backpropagation steps.

The batch size determines the maximum limit of how many items can be processed in parallel concurrently, as TensorFlow will dispatch work for the entire batch and wait before performing backpropagation. If the total number of items in a training dataset is much greater than the number of items in a batch and the computational cost of each item is much smaller than the capabilities of the underlying hardware, then it can be assumed that the size of a batch may limit the speed at which the entire dataset is processed; for small batch sizes, the hardware may be able to process the entire batch in parallel but have unutilized computing elements that a larger batch would be able to use. Studies observe this phenomenon, noting that increasing the batch size can yield greater time efficiency on parallel systems [4][5]. Liu et al. focus on dynamically changing the batch size while training to improve the efficiency of the training process (loss improvement over time) [4]. Ramirez-Gargallo et al. also observe the same increase in performance as both batch size and thread counts increase [5]. Increasing the batch size may decrease accuracy though, as the weights of the network are not adjusted as often over the course of training as observed by Liu et al. In this work, a sample CNN is created and run on two different computer systems with varying parameters to evaluate the impact of changing the batch size during training.

III. Experimental Setup

The neural network used for testing the impact of batch size is a CNN designed to classify images from the MNIST database of handwritten digits. The structure of the network can be seen in Figure 2, with a 28x28 input layer for each pixel of the input images, followed by two alternating layers of convolutions and pooling with a kernel size of 4x4 and a pooling size of 2x2. After flattening the network incorporates two alternating layers of dropout and densely connected neurons, the first dense layer having 500 neurons and the final output layer having 10 neurons corresponding to each digit.

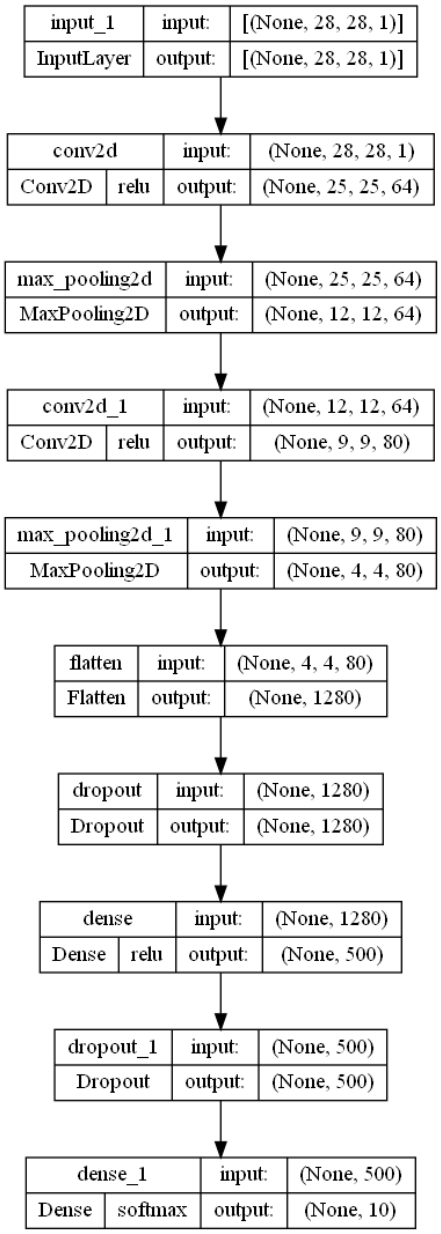

Figure 2. Structure of the CNN used

Each convolution and dense layer is set to use rectified linear unit (ReLU) activation except the final layer which was set to use softmax activation to fit with the purpose of classification and generate a set of percentages for each digit to determine which it appears most like to the network. All weights and bias are initialized using TensorFlow's RandomNormal initializer and all seeds for random number generation are set to 1000 to ensure uniform initialization between runs.

The Python script implementing the network is run on two systems, a desktop system with a Ryzen 1800X CPU and Radeon RX 6800XT GPU, and Wichita State University's BeoShock system using two Intel Xeon Gold 6240 CPUs and a Nvidia Tesla V100 GPU. The desktop is a standalone system; when the CNN script is run on the desktop system, there is no other CPU or GPU intensive tasks running. The desktop system is later upgraded to a Ryzen 5700X CPU and the same tests are run again. The BeoShock is a shared high-performance computing (HPC) system; the CNN script is run using the Slurm scheduling. More details on the hardware of each system are given in Table 1, including memory capacities, core counts, and nominal clock frequencies for each system.

TABLE I. SYSTEM HARDWARE

Component	System 1 (Desktop)	System 2 (HPC)
CPU	Ryzen 1800X @ 3.6GHz	2x Intel Xeon Gold 6240 @ 2.6GHz
	Ryzen 5700X @ 4 GHz	
CPU Cores / Threads	8C / 16T	2x 18C / 36T
Memory	32 GB @ 2400MHz	384 GB
GPU	Radeon RX 6800 XT @ 1825MHz	Tesla V100 @ 1245 MHz
GPU Cores	4608	5120

IV. SIMULATION RESULTS

In this section, we present the total training time and final accuracy obtained by running the CNN program on the described (i) desktop system with Ryzen 1800X, (ii) desktop system with Ryzen 5700X, and (iii) BeoShock system for various batch sizes and epochs.

A. CNN on Desktop: CPU Only

Figures 3 and 4 show the training time and accuracy, respectively, when CNN is run using only the system's CPU (Ryzen 1800X at 3.6 GHz). As illustrated in Figure 3, the training time remains almost the same as the batch size increases. While the training time shows a small decrease with larger batch sizes, it is much shallower.

According to Figure 4, the accuracy is somewhat decreased with larger batch sizes, but remains above 90%. The training time overall is also much larger compared to the GPU.

We repeat the CNN tests on the desktop system with the newer Ryzen 5700X processor. As shown in Figure 5, we notice a significant improvement in training time but the same patterns emerge as the previous tests; increasing the batch size only provides marginal improvements at relatively small numbers and plateaus quickly with no performance gain.

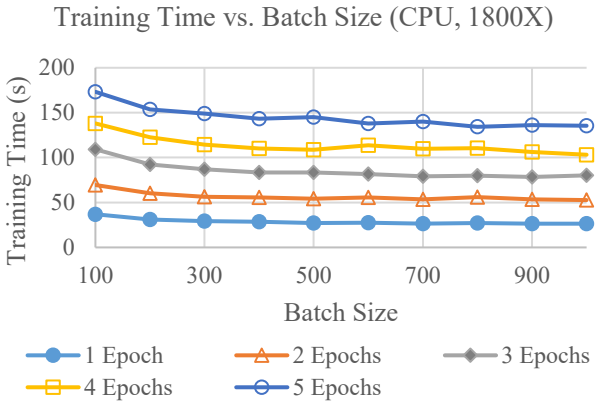

Figure 3. Training time for the desktop with Ryzen 1800X CPU

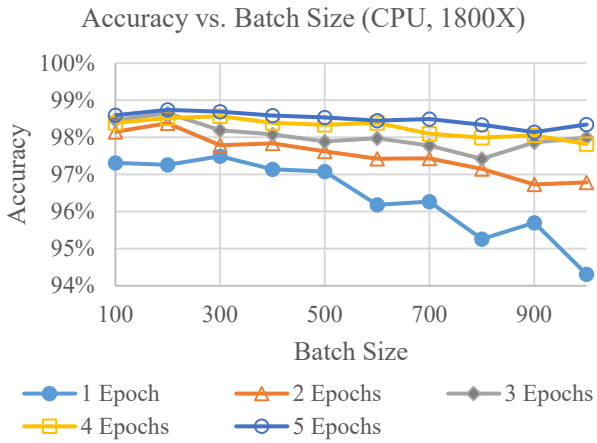

Figure 4. Accuracy for the desktop with Ryzen 1800X CPU

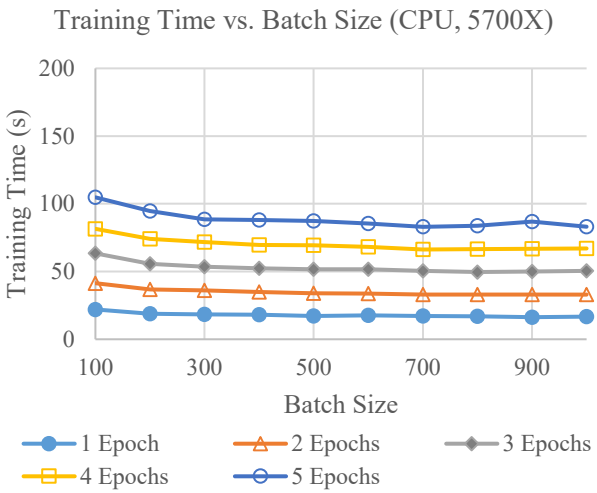

Figure 5. Training time for the desktop with Ryzen 5700X CPU

The accuracy plot in Figure 6 again shows similar behavior to previous tests, with accuracy rising with the number of epochs. This behavior seems to be consistent across all variations, with larger batch sizes reducing accuracy when the number of epochs is small, but rising with the number of epochs regardless of the batch size until an upper limit is reached.

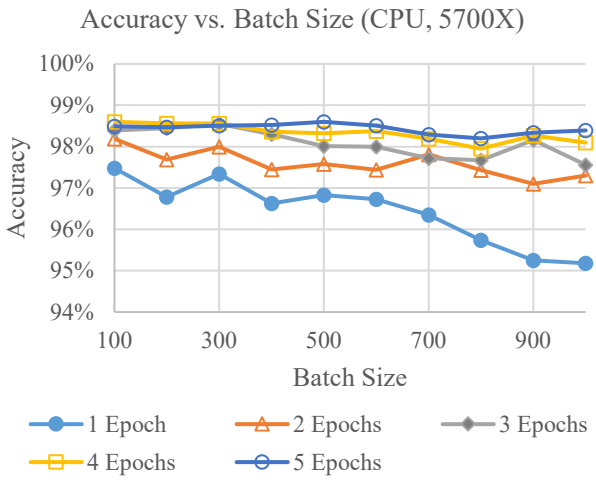

Figure 6. Accuracy for the desktop with Ryzen 5700X CPU

B. *CNN on Desktop: GPU Only*

Figures 7 and 8 show the training time and accuracy, respectively, when CNN is run using only the system's GPU. The training time shows a significant decrease as the batch size increases, indicating that the GPU can process more items in parallel than the initial batch size value would allow, tapering off around 800 items. The training time also scales linearly with the number of epochs, as each epoch requires approximately the same amount of time to process the entire dataset. The performance improvement with larger batch sizes in this case is so significant that it is faster to process 5 epochs with a batch size of 1000 than it is to process a single epoch with a batch size of 100.

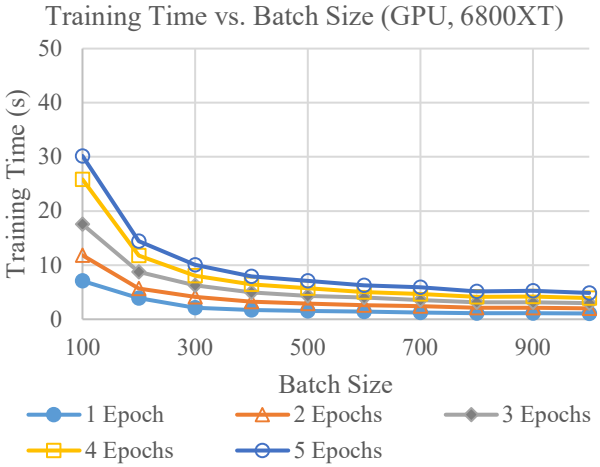

Figure 7. Training time for the desktop with 6800XT GPU

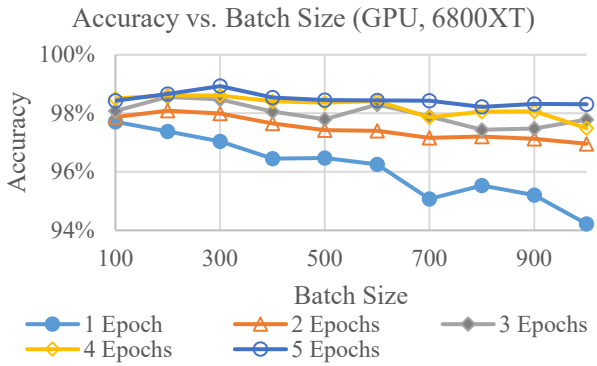

Figure 8. Accuracy for the desktop with 6800XT GPU

The accuracy is somewhat decreased with larger batch sizes, but remains above 94% even for one epoch and quickly improves to near 99% for five epochs.

The total GPU usages is captured during training using AMD's Adrenaline management software and can be seen in Figure 9. The pre- and post-training periods show a background level of GPU usage, while the training period shows spikes above 50% as the neural network trains. As the batch size increases, the utilization also increases while the period of each test decreases.

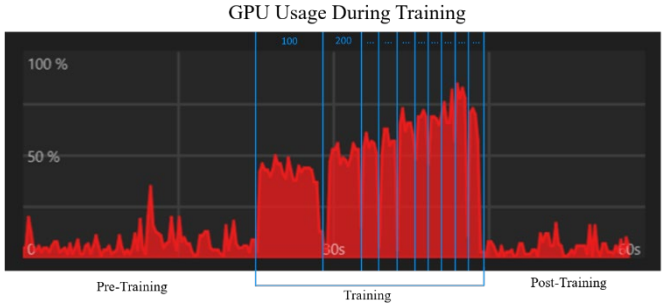

Figure 9. Training time for the desktop with 6800XT GPU

C. *CNN on BeoShock*

The CNN program was also run on the BeoShock system using the Slurm scheduler, producing very different results as seen in Figures 10 and 11.

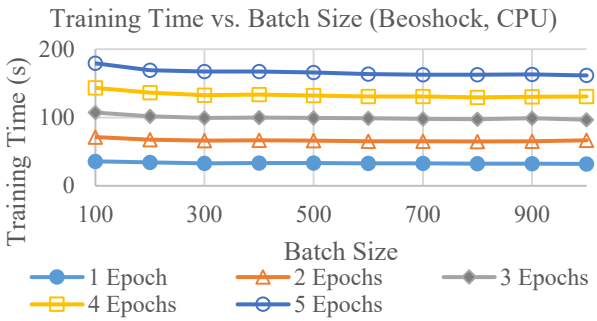

Figure 10. Training time for the BeoShock system using CPU

Accuracy for the BeoShock trials showed the same trends as with the desktop, decreasing with higher batch sizes but recovering with more epochs.

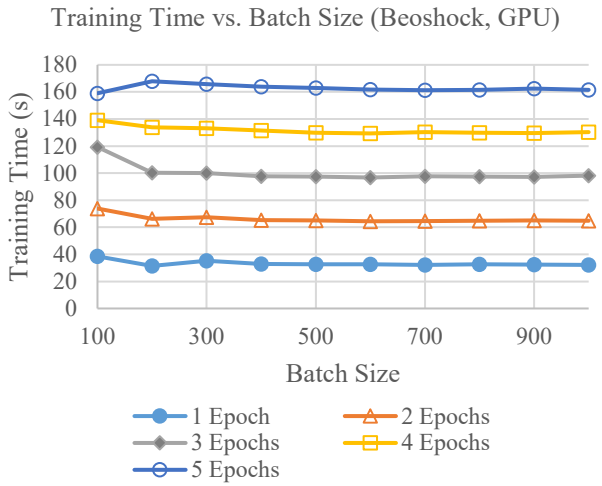

Figure 11. Training time for the BeoShock system using GPU

Unlike the desktop system, for BeoShock, both the CPU and GPU tests show no significant improvement with larger batch sizes. The GPU run took significantly longer than its equivalent on the desktop system (see Figures 7 and 11). A possible explanation for this large difference is that BeoShock is a shared system with different users running concurrent HPC tasks, while the desktop was a single-user system only running background tasks, meaning the BeoShock system would have a much lower limit on the maximum effective computational throughput and would be more easily saturated with work by a smaller batch size. This is a good reminder that optimal performance depends on the conditions of the environment the program is running in.

V. Conclusion

The impact of batch size on the training time and accuracy of TensorFlow-based CNNs is a crucial consideration that can significantly influence the overall model performance. This study investigates the impact of the batch size on the performance of a CNN. The CNN is employed for image classification using the MNIST database of handwritten digits. The experiments are conducted on two computing platforms: a desktop equipped with eight CPU cores and 4608 GPU cores, and an HPC system featuring two sets of 18 CPU cores and 5120 GPU cores. Through the exploration of various batch sizes and epochs, it becomes evident that choosing an appropriate batch size is a balancing act, influenced by factors such as model architecture and dataset characteristics. This is because the method TensorFlow uses to schedule the computational workload. Simulation results suggest that larger batch sizes may reduce training time with negligible or recoverable losses in accuracy. However, the environment a CNN is running in can affect the point at which it effectively saturates the computing system with work. Therefore, optimization must be taken on a case-by-case basis to determine the best parameters. In particular, these effects are observed on a small-scale dedicated system and larger multi-computer systems or shared computing systems may not see the benefits of this tuning.

References

[1] M. Abadi, A. Agarwal, P. Barham, et al., “TensorFlow: Large-scale machine learning on heterogeneous distributed systems,” TensorFlow, 2015. http://download.tensorflow.org/paper/whitepaper2015.pdf

[2] M. Ramchandani, H. Khandare, P. Singh, et al., “Survey: TensorFlow in Machine Learning,” Journal of Physics: Conference Series, vol. 2273, no. 1, May 2022.

[3] A. Gholami, A. Azad, P. Jin, K. Keutzer, and A. Buluc, “Integrated Model, Batch, and Domain Parallelism in Training Neural Networks,” in Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 77–86, July 2018. https://doi.org/10.1145/3210377.3210394

[4] B. Liu, W. Shen, P. Li, and X. Zhu, “Accelerate Mini-batch Machine Learning Training With Dynamic Batch Size Fitting,” in International Joint Conference on Neural Networks (IJCNN), pp. 1-8, Budapest, Hungary, 2019. doi: 10.1109/IJCNN.2019.8851944

[5] G. Ramirez-Gargallo, M. Garcia-Gasulla, and F. Mantovani, “TensorFlow on State-of-the-Art HPC Clusters: A Machine Learning use Case,” in IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 526-533, Larnaca, Cyprus, 2019. doi: 10.1109/CCGRID.2019.00067