Deep Learning with NVIDIA DGX-1: Practical Considerations for the End User

WWT Artificial Intelligence Research & Development

MARCH | 2018

World Wide Technology
wwt.com

Table of Contents

Abstract

The NVIDIA DGX-1 is a state-of-the-art integrated system for deep learning and Artificial Intelligence (AI) development. Making use of eight interconnected NVIDIA Tesla V100 graphics processing units (GPUs), the DGX-1 offers dramatic acceleration of deep learning algorithms over central processing unit (CPU)-based hardware. In this paper, we highlight a few best practices that enable the DGX-1 end-user to fully capitalize on its industry-leading performance. Benchmark testing was conducted with a common GPU workload, convolutional neural network (CNN) training, using the Keras Deep Learning API. We first examined the dependence of training efficiency on three factors: batch size, input image size and model complexity. Next, the scalability of training speed was assessed using a multi-tower, data-parallel approach. Finally, we demonstrate the importance of learning rate scaling when employing multiple GPU workers.

Business Justification

GPUs are driving the present AI boom. The value of GPUs in the AI domain lies in their ability to compute complex mathematical operations orders of magnitude faster than CPUs. Deep neural networks (DNNs) are a class of algorithms to which the processing prowess of GPUs is particularly well suited. Ever since a DNN and GPU were used to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012,[1] the field of machine learning has seen an explosion of new DNN variants aimed at solving the field's most exciting and difficult problems, e.g., driverless vehicles and human-machine dialog. Given the growing complexity of these algorithms and the ever-increasing size of the unstructured data required to run them, GPUs have become the cost of entry for AI development.

The NVIDIA DGX-1 is the first off-the-shelf hardware/software stack specifically designed for DNN workloads. The unit is composed of eight topologically optimized NVIDIA Tesla V100 GPUs with 16 GB of RAM each, two Intel Xeon E5-2698 v4 CPUs, 512 MB system RAM, 4x1.92 TB of SSD storage and 4x100 Gb/s Infiniband network adapters and occupies about as much space as three compute nodes in a server rack. All of this hardware comes pre-loaded with NVIDIA's software stack, including a deep learning SDK that optimizes DL computations for single- and multi-GPU processing. Additionally, the device supports all major DL programming frameworks through NVIDIA-Docker, a GPU-enabled version of the popular Docker containerization engine. The DGX-1 is preconfigured as such to get the end-user up and running in as little time as possible and with maximum flexibility.

While the DGX-1 is the current industry leader in deep learning, NVIDIA has been boosting the performance of its GPU devices over the last few years at a staggering pace. Compared to the K80 GPU Accelerator, which was the state of the art in 2014, a single Tesla V100 offers a nearly 7X speedup. In fact, a recent hardware and software upgrade for the DGX-1 netted a 3X single-GPU speedup compared to the original configuration.[2] Used in concert, the eight GPUs of the DGX-1 provide enough compute power to handle the most intensive of deep learning workloads. However, this industry-leading performance is largely dependent on the specific routines and parameters that are defined by the end-user. Herein, we describe some points of consideration for getting the most out of a DGX-1 installation.

Experimental Setup and Methodology

CNN MODELS

Three convolutional neural network (CNNs) architectures were used for comparative performance analysis: ResNet-50[3], VGG16[4], and a shallow network similar to LeNet-5.[5] The shallow network, termed LittleCNN, was composed of two convolutional blocks of 32 and 64 output volume depth, a fully connected layer of 512 neurons, and a final 10-node softmax layer. Each convolutional block consisted of two successive 3x3 convolutions followed by 2x2 max pooling. The ReLU activation function was used for all layers and dropout regularization was applied after the first and second convolutional blocks and after the fully connected layer at drop rates of 0.25, 0.25, and 0.5, respectively. All models were trained using RMSprop optimization with a learning rate of $1x10^{-4}$, unless otherwise noted.

DATASETS

The CNN models were trained on the CIFAR-10 dataset as well as synthetic image data. CIFAR-10 consists of 60,000 32x32 color images of 10 different classes and is split into 50,000 training and 10,000 validation images.[6] CIFAR-10 images were normalized to values between 0 and 1 prior to forward pass through the network. A synthetic image set was constructed by generating 1,024 224x224 arrays of values randomly sampled from a uniform distribution over [0,1]. A corresponding label set was created by randomly assigning each synthetic image a one-hot encoded integer value of 0 through 9. Synthetic images were used to emulate the larger image sizes, e.g., ImageNet, that are typically fed to deep CNNs like ResNet and VGG variants. Because these images were randomly generated, they were used to evaluate GPU performance purely on computational speed without regard to classification accuracy.

HARDWARE AND SOFTWARE

All experiments were conducted on a NVIDIA DGX-1 (8xTesla V100) running Ubuntu 16.04.3. Model training was carried out using Keras 2.1.5 with TensorFlow 1.7.0 as the backend. A GPUenabled Keras Docker file was obtained from GitHub7, which included TensorFlow and CUDA/ cuDNN dependencies.

MULTI-GPU TRAINING

Multi-GPU training was performed using the *multi_gpu_model()* function included with Keras, which accommodates multi-tower, data-parallel training. The data-parallel approach is outlined as follows:

1. Instantiate CNN model on CPU, where the CPU represents the "parameter server."
2. Instantiate a copy of the model on *n* devices (GPUs). Copied model instances are also referred to as towers.
3. During training, split the input batch into *n* equal-sized minibatches, where *n* is the number of GPUs or towers.
4. Compute forward passes for each sample in the respective minibatch on each tower simultaneously.
5. Compute backward pass on each tower simultaneously to obtain gradients for each model copy.
6. Once gradients have been computed for all towers, concatenate gradients on CPU, update model weights accordingly and pass new weights from the parameter server to the GPU devices.
7. Repeat steps 3-6

Data parallelism allows a higher throughput of images during training, and therefore reduced training time. It should be noted that the above implementation is an example of synchronous training, where the parameter server (CPU) must wait for all towers (GPUs) to complete backpropagation before updating the model weights. Consequently, a truly linear scaleup of training speed is theoretically impossible. Currently, Keras only supports synchronous training with a multi-tower approach. The effect of synchronicity on multi-GPU training efficiency is discussed further in the *Results* section of this paper.

BENCHMARKING EXPERIMENTS

The following investigations were carried out using a single-GPU training scheme.

a) *Batch size.* The effect of input image batch size on image throughput, GPU utilization and validation accuracy was evaluated by training LittleCNN on CIFAR-10 images for 30 epochs with a varying batch size of 16, 32, 64, 128, 256, 512, 1024, and 2048 images.
b) *Image size.* Experiment a) was repeated (up to a batch size of 256) using large 224x224 synthetic images to examine the effect of input image dimensions on image throughput and GPU utilization.
c) *Model size.* Large CNNs ResNet-50 and VGG16 were trained for 30 epochs with synthetic images (224x224) using a batch size of 32. GPU utilization was monitored and compared to the results of experiment b) to assess the effect of neural network complexity on training efficiency.

The multi-GPU training scheme described above was implemented for the following.

d) *Multi-GPU acceleration.* The relationship between image throughput and number of GPUs used was investigated by training LittleCNN, ResNet-50 and VGG16 on synthetic images (batch size = 32) using 1, 2, 4 and 8 of the available GPUs in the DGX-1.

e) *Learning Rate Scaling.* LittleCNN was trained on CIFAR-10 for 30 epochs using 1, 2, 4, and 8 GPUs with and without adjustment of the learning rate according to the number of GPU workers (batch size = 128 images per GPU). A constant learning rate of 1×10^{-4} was used for all tests, then the tests were repeated using a learning rate of $1 \times 10^{-4} \cdot N$, where N is the number of GPU workers. Time-dependent validation accuracy was monitored for all tests to evaluate the effect of learning rate scaling on model convergence.

Results

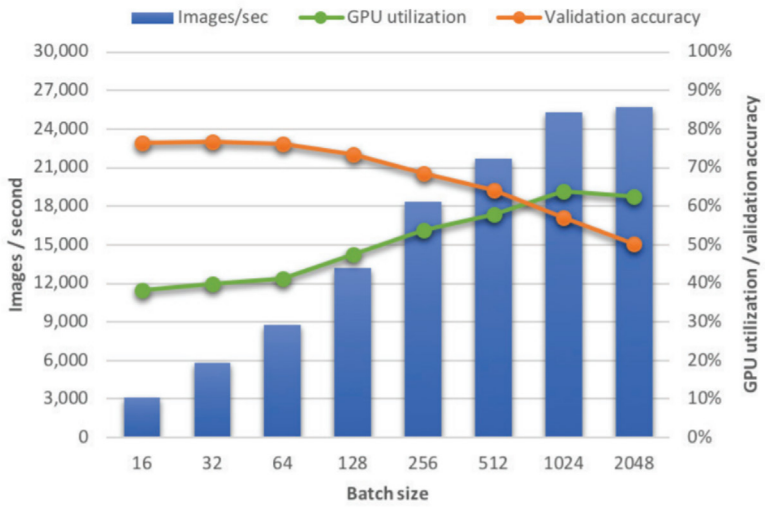
BATCH SIZE

The processing efficiency of a GPU during deep neural network training is dependent on several factors. We first examined the effect of batch size on training speed. Batch size refers to the number of input samples, images in this case, that are fed through the network during each weight update step. In addition to training speed, batch size can have a marked effect on the learning capacity of a neural network. If the input batch is too small, the gradients calculated during each training step can be so noisy that the model has difficulty learning anything at all. If it is too large, convergence tends to be slow, and one runs the risk of converging on a local minimum instead of pushing to the globally optimal solution. Assuming a total image set size of at least a couple thousand images, a batch size of 32 to 128 images typically provides a good compromise between these two extremes.

The LittleCNN network was trained on CIFAR-10 for 30 epochs using a single GPU with an input batch size ranging from 16 to 2048 images. The combination of the shallow network architecture of LittleCNN and the small size of the CIFAR-10 images results in substantially decreased computational demand compared to a modern CNN workflow. Consequently, a very large input batch size is needed to maximize training throughput (Figure 1). We observed that image throughput only began to level off at around 26,000 images per second once a batch size of 1024 images was used. Using a batch size of 16 images, the network was only able to process 3,000 images per second. This discrepancy illustrates a phenomenon known as GPU starvation, where the GPU spends some amount of time in an idle state during training due to a bottleneck in the input pipeline. In this case, because the images are small, the GPU computation time required to process a small batch of images is less than that of overhead processes. For example, a batch of input images must be copied from the CPU to the GPU device during each training iteration. When the batch size is small, the task of copying a batch to GPU becomes an input bottleneck, which results in underutilization of the GPU. This effect is reflected in the average GPU utilization curve shown in Figure 1.

FIGURE 1:

Dependence of image throughput, average GPU utilization and validation accuracy on batch size for LittleCNN trained on CIFAR-10 using a single GPU. Validation accuracy represents model classification performance after training for 30 epochs.



Batch size also has a marked effect on the convergence rate of the model. Figure 1 shows the validation accuracy on the test image set after 30 training epochs. Using a batch size of 2048, the final accuracy is only 50%, while a batch size of only 16 images achieves an accuracy of 76%. This occurs because a larger batch size affords fewer weight updates during a training epoch, and therefore, the model learns more slowly. In practice, large batch sizes on the order of 1x103 are typically not used unless a distributed training approach is implemented. This approach is discussed in the context of multi-GPU data parallelism later in this paper.

IMAGE SIZE

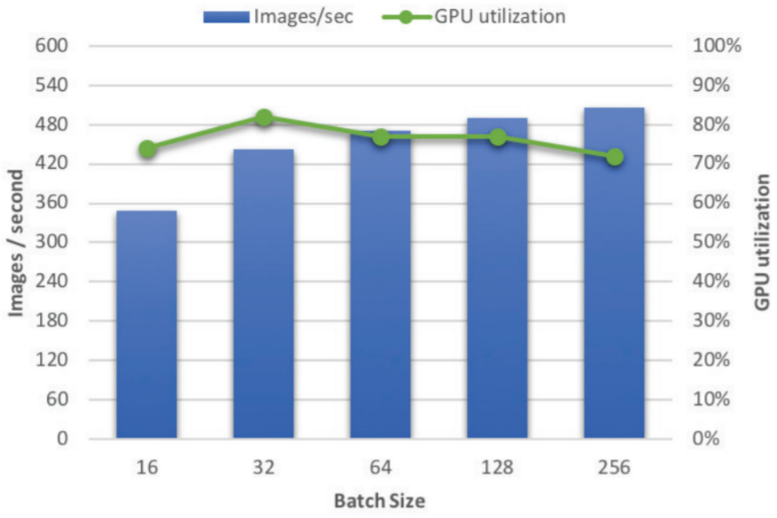
We have demonstrated that underutilization of the GPU during training can be corrected by increasing the batch size, which increases the number of computations the GPU must complete during each training step. Another way to raise the computational demand on the GPU is to increase the size of the input images. To examine the effect of image size on throughput and GPU efficiency, we repeated the experiment above using much larger 224x224-pixel synthetic images (49X the pixel count of CIFAR-10). Figure 2 shows the training speed in images per second and GPU utilization for LittleCNN trained on the large synthetic images using batch sizes of 16 to 256 images, above which the GPU memory became a constraint.

As anticipated, training velocities using synthetic images are significantly lower than those observed for the CIFAR-10 experiments. We can expect image throughput with larger images to be lower, since the time it takes to process an image will always be greater regardless of GPU optimization. More notably, however, is the much lesser degree to which the batch size affects throughput. Using large images, the training speed is near its

maximum with a batch size of 64 images, compared to 1,024 when the model was trained on CIFAR-10. Furthermore the relative speedup between 64 and 16 images per batch was only 1.3X when using synthetic images, while the corresponding rate increase for CIFAR-10 was 2.8X. Additionally, GPU utilization had no apparent dependence on batch size and was always above 70% when the model was trained with synthetic images, indicating that the GPU was sufficiently fed in all cases.

FIGURE 2:

Dependence of image throughput and average GPU utilization on batch size for LittleCNN trained on 224x224 synthetic images for 30 epochs using a single GPU.



MODEL SIZE

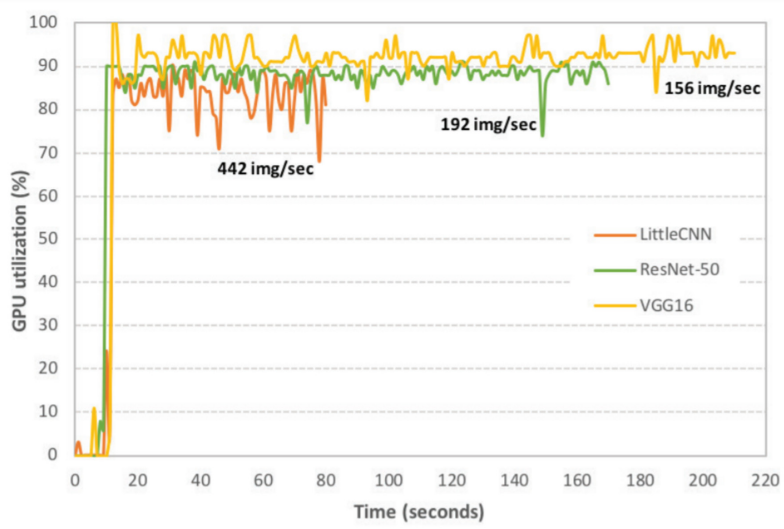
In addition to batch size and input image size, the architecture of the deep neural network itself is directly related to computational demand, and therefore, image throughput. The size of a CNN is typically described in terms of depth, where depth refers to the number of convolutional plus terminal fully connected layers in the network. Since the introduction of AlexNet, CNNs have grown deeper to yield incrementally greater classification accuracy on diverse datasets, such as ImageNet. For comparison, AlexNet (2012) contains seven layers and VGG16 (2014) contains 16. The greater depth of modern CNNs as well as larger image sizes used for classification problems have made GPUs indispensable, as training these networks on CPU is prohibitively slow. In order to assess the performance of the DGX-1 on a more modern workload, we repeated the previous experiment using two additional moderately deep CNNs: ResNet-50 and VGG16.

Image throughput and GPU utilization for LittleCNN, ResNet-50 and VGG16 are illustrated in Figure 3. Each network was trained on a single GPU for 30 epochs with a batch size of 32, as a 64-image batch caused the GPU to run out of memory when training the larger ResNet-50 and VGG16 models. As expected, the training speeds of the larger networks are less than that of LittleCNN (442 images/second). We found the average

image throughput for ResNet-50 and VGG16 to be 192 and 156 images per second, which is consistent with the DGX-1 benchmarking results published on the TensorFlow website: 195 and 144 images per second, respectively, using a single GPU, synthetic 224x224 images and a batch size of 32.[8] Interestingly, the three networks are also differentiable on the basis of GPU utilization. LittleCNN was the least efficient, with an average GPU usage of 82%, while ResNet-50 and VGG16 showed average utilizations of 88 and 92%, respectively. Thus, there appears to be a correlation between network complexity and GPU efficiency. While this makes sense intuitively, the exact nature of this relationship is unclear. Since we are operating well outside the regime of GPU starvation, one might expect GPU utilization to approach the theoretical maximum regardless of the specific network architecture, but empirically, this is not the case. The authors of TensorFlow suggest a full utilization criterion of 80-100%, within which input pipeline bottlenecks can effectively be ruled out.[9] This rather large window suggests an inherent variability in GPU utilization during training. Therefore, we can infer that all three models achieved full utilization despite their relative differences in measured GPU usage.

FIGURE 3:

Time-dependent GPU utilization for LittleCNN, ResNet-50 and VGG16 network architectures trained for 30 epochs on 224x224 synthetic images with a batch size of 32.



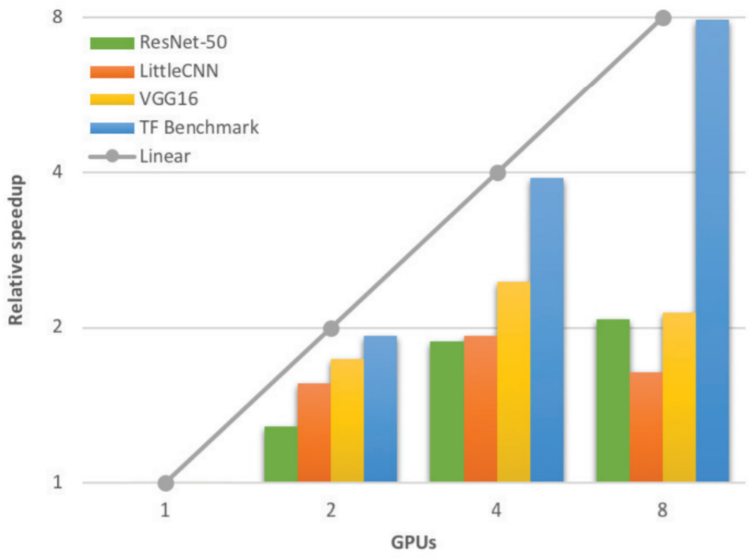
MULTI-GPU ACCELERATION

We next investigated training speed enhancement using a multi-GPU approach on the DGX-1. Multi-GPU training implies using two or more of the GPU devices in parallel with the aim of increasing image throughput and consequently reducing the time required to achieve a desired level of classification accuracy. The most common form of GPU parallelization is called data parallelism, which is detailed in the *Experimental Setup and Methodology* section of this paper. The utility of this approach has been described recently by researchers at Facebook, who were able to achieve state-of-the-art accuracy on the ImageNet dataset in one hour using 256 GPUs in parallel.[10] Given that training the same model on the same dataset using eight equivalent GPUs required 29 hours, Facebook's large-scale results represent a near linear scaleup in terms of training speed. We attempted to replicate these findings at much smaller scale using 1-8 GPUs on the DGX-1.

We trained the three models discussed previously on synthetic images using 1-8 GPUs with a batch size of 32 x G, where G is the number of GPU devices used for training. Figure 4 shows the relative training speed enhancement with respect to image throughput as a function of GPUs. Doubling the number of GPUs from one to two provided respectable enhancement for the VGG16 model, with a speedup factor of 1.74, while ResNet-50 showed only 1.29x improvement. However, a significant discrepancy relative to linear scaling became apparent when more than 2 GPUs were used. With four GPU towers, the maximum scale factor achieved in our experiments was 2.46 with VGG16, while both ResNet-50 and LittleCNN showed speedup factors less than 50% of linear. Furthermore, scaling from four to eight GPUs offered only marginal improvement for ResNet-50 and actually showed a decrease in throughput for the LittleCNN and VGG16 networks.

FIGURE 4:

Image throughput speedup with multi-GPU training relative to a single GPU. All experiments were conducted using 224x224 synthetic images with a batch size of 32 images per GPU. TensorFlow benchmark results were generated with the ResNet-50 network architecture.



Our multi-GPU benchmarking results highlight a limitation in distributed CNN training caused by synchronous stochastic gradient descent (SGD). In synchronous SGD, model gradients are computed locally on each GPU device then copied to the parameter server where the gradients are averaged to obtain the new global updated weights. The updated model parameters are then copied to each GPU worker to be used for the next round of forward passes. While this method of gradient averaging over a split input batch has been shown to be roughly equivalent to single-GPU training on the whole batch in terms of model accuracy, it has some drawbacks with respect to training speed. Most notably, there is an inherent penalty on GPU efficiency when training is conducted in a synchronous fashion. Because computation of the new model parameters is dependent on the results from all GPU towers, all GPUs are required to complete training on their respective input batches before a model update can be implemented. Consequently, the time required to complete a single model update is constrained by the slowest GPU worker. This effect becomes more apparent as more GPUs are added to the training job, since total GPU idle time will be proportional to the total number of GPUs. Furthermore, network bottlenecks between the parameter server and GPU workers can exacerbate this inefficiency. While network constraints should not apply to the DGX-1, which is self-contained, they must be considered when training is distributed across multiple servers.

The drawbacks of synchronous GPU training can be overcome by using an asynchronous SGD (ASGD) training scheme. In ASGD, the global model parameters stored on the parameter server are updated by each GPU worker independently as soon as local gradient computation has completed. This eliminates the overhead time penalty associated with synchronous updates. ASGD can be implemented with native TensorFlow

primitives to achieve near linear multi-GPU scaleup. A benchmark performance summary published by TensorFlow using asynchronous multi-GPU training on the DGX-1 is given in Figure 4. ASGD has clear benefits in terms of training throughput and scalability in multi-GPU environments, but it is not without its own drawbacks. Because the global parameters are modified each time a GPU worker completes a training iteration regardless of that individual worker's progress relative to others, the shared model can potentially be reverted to a less optimized state by a single lagging GPU. This is known as the "stale gradient" problem. Stale gradients can slow convergence to a globally optimal solution, which means that asynchronous training tends to require more epochs of training to achieve a given level of accuracy compared to a synchronous implementation.

LEARNING RATE SCALING

In addition to computational efficiency, one must consider the impact on training effectiveness when employing a multi-GPU architecture. Training throughput is only part of the picture. It is equally important to examine how quickly a model learns when scaleup is implemented. As described previously in the context of single-GPU training, hyperparameters like batch size can have a substantial effect on model convergence. In the case of multi-GPU training, it has been demonstrated that adjustment of the learning rate is important to achieve optimal performance.[10] Learning rate is a user-specified value that determines the magnitude by which model weights are adjusted during each training step. Its effect on the updated model weights can be described by

$$\Delta W_t = \alpha \cdot \nabla L(y, F(x; W_t))$$

where ΔW_t is the change in model weights for iteration t, α is the learning rate, and the rightmost expression describes the computed loss gradient as a function of the current model weights. Thus, increasing the learning rate increases the magnitude of change in the model weights. The learning rate must be high enough such that training does not converge in a local optimum, but low enough that the global optimum solution is not passed over.

It has been shown that linear scaling of the learning rate with respect to the number of GPU workers improves the accuracy of multi-GPU implementations. To test this, we conducted 30 epochs of training using CIFAR-10 and LittleCNN on one, two, four and eight GPUs with unscaled and scaled learning rates. Figure 5 shows the time-dependent cross-entropy loss on the validation images using a constant learning rate of 1×10^{-4} for each GPU configuration. There is a clear incremental performance loss as the number of GPUs is increased. However, when the learning rates were scaled to account for the number of

GPUs, validation losses were roughly equivalent after 30 training epochs (Figure 6). This effect can be explained on the basis of relative batch size. Because the number of images sent to each GPU during training is fixed at 128, increasing the number of GPU workers effectively increases the total batch size for each training step. Consequently, a single training epoch with eight GPUs includes 8x fewer weight updates compared to one epoch using a single GPU. This same phenomenon was observed while examining the effects of batch size on training speed and accuracy using one GPU (Figure 1): larger batch sizes imply fewer learning opportunities per epoch. As illustrated in Figure 6, this effect can be reversed simply by increasing the magnitude of learning at each of those less frequent update steps, i.e., increasing the learning rate.

FIGURE 5:

Time-dependent cross-entropy loss for multi-GPU training of LittleCNN on CIFAR-10 with a batch size of 128 images per GPU and a constant learning rate of $1x10^{-4}$.

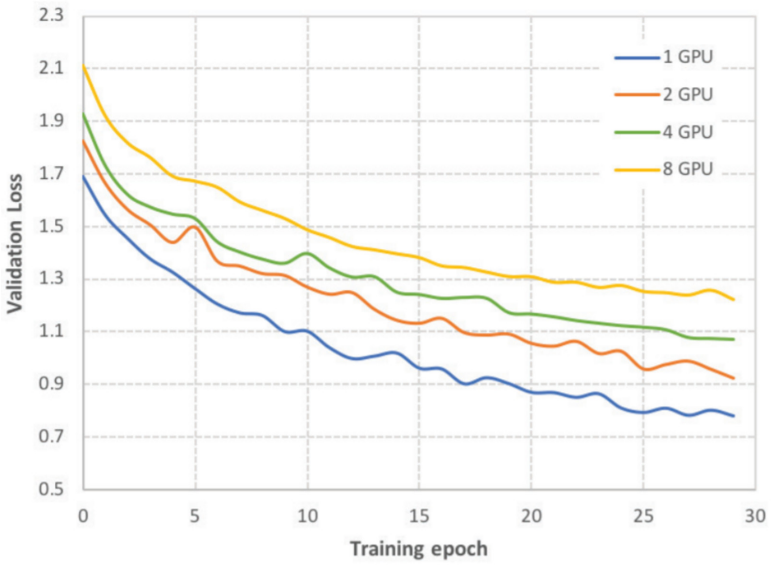
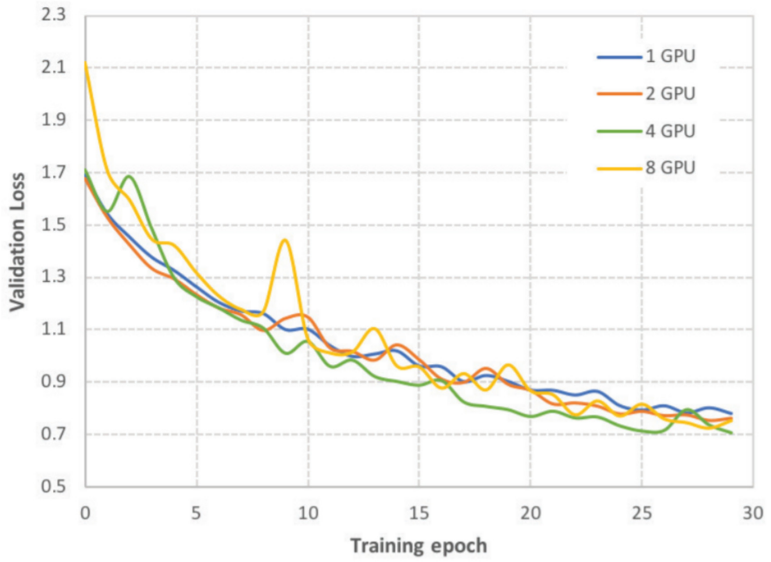


FIGURE 6:

Time-dependent cross-entropy loss for multi-GPU training of LittleCNN on CIFAR-10 with a batch size of 128 images per GPU and scaled learning rates of $1x10^{-4}$, $2x10^{-4}$, $4x10^{-4}$, $8x10^{-4}$ for 1,2,4 and 8 GPUs, respectively.



Conclusion

We have highlighted some practical considerations for the deep learning practitioner relevant to neural network training on the NVIDIA DGX-1. Benchmarking experiments showed that GPU performance is related to three dimensions: 1) the size of the input batch, 2) the size of the input images, and 3) the size of the neural network architecture. We demonstrated that the enhanced processing power of GPUs can only be fully realized if the computational demand on the GPU during training exceeds all other overhead processes, such as the I/O pipeline. All three of the factors listed above are proportional to the number of computations that must be performed on the GPU during training, and thus, directly impact GPU utilization. Large CNN models such as ResNet-50 and VGG variants with input images of at least 224x224 pixels show complete utilization with as few as 32 images per batch. However, shallower models, such as those used to solve much smaller image sets like CIFAR-10 and MNIST (32x32 and 28x28 pixels, respectively), will exhibit GPU starvation at moderate batch sizes. In our testing, a simple three-layer CNN trained on CIFAR-10 achieved maximum utilization only when a batch size of at least 2048 images was used.

We also examined the scalability of distributed training using a data-parallel approach with Keras and TensorFlow. We found that while implementation with Keras provided reasonable scaleup using two GPUs compared to one, increasing the number of GPUs beyond two showed sharply diminishing returns. This is a well-documented phenomenon that occurs during synchronous training with multiple GPU workers. Currently, Keras only supports synchronous SGD training. However, near linear scaleup can be accomplished using low-level TensorFlow primitives for asynchronous SGD, as established by TensorFlow's published benchmarks on the DGX-1. Finally, we demonstrated the importance of learning rate adjustment when using multiple GPUs. It is recommended to increase the learning rate by a factor of N for a data-parallel training scheme, where N is equal to the number of GPUs. We found that this linear scaling criterion rectified depressed model convergence rates caused by multi-GPU training.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 2012 1: 1097-1105.
2. https://developer.nvidia.com/cudnn.
3. He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition 2015.
4. Simonyan, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition. ICLR conference paper 2015.
5. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. Gradien-based learning applied to document recognition. Proc. Of the IEEE 1998.
6. https://www.cs.toronto.edu/~kriz/cifar.html.
7. https://github.com/keras-team/keras/tree/master/docker.
8. https://www.tensorflow.org/performance/benchmarks.
9. https://www.tensorflow.org/performance/performance_guide.
10. Goyal, P., Dollar, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tullock, A., Jia, Y., He, K. Accurate, large minibatch SGD: training ImageNet in 1 hour 2017.

ARTIFICIAL INTELLIGENCE
RESEARCH & DEVELOPMENT

Learn more about WWT's Artificial Intelligence Research & Development at:
https://www.wwt.com/artificial-intelligence-research-and-development

ABOUT WWT ARTIFICIAL INTELLIGENCE RESEARCH & DEVELOPMENT

The Artificial Intelligence Research & Development program at WWT is an applied research initiative focused on investigating the one to three-year horizon of the artificial intelligence space. The AI R&D program conducts internal projects grounded in WWT's deep understanding of industry use cases and produces reusable components and white papers to share with customers and the AI community.

Through strategic partnerships, cutting-edge data science and ML Ops skills, and ability to test and learn in the Advanced Technology Center and public cloud, the AI R&D team advances WWT's knowledge of the AI and ML space, thus allowing WWT to be on the bleeding edge and remain strategic advisors to businesses in their AI needs.

ABOUT WWT

Founded in 1990, WWT has grown to become a global technology solution provider with nearly $12 billion in annual revenue. With thousands of IT engineers, hundreds of application developers and unmatched labs for testing and deploying technology at scale, WWT helps customers bridge the gap between IT and business. By bridging leading technology companies together in a physical yet virtualized environment through its Advanced Technology Center, WWT integrates individually impressive technologies to produce game-changing solutions.

Based in St. Louis, WWT employs more than 6,000 employees and operates over 4 million square feet of warehousing, distribution and integration space in more than 20 facilities throughout the world.