EPJ Web of Conferences **214**, 06016 (2019) https://doi.org/10.1051/epjconf/201921406016
CHEP 2018

Scaling studies for deep learning in Liquid Argon Time Projection Chamber event classification

Jan Strube[1,2,*], *Kolahal* Bhattacharya[1], *Eric* Church[1], *Jeff* Daily[1], *Malachi* Schram[1], *Charles* Siegel[1], and *Kevin* Wierman[1]

[1]902 Battelle Boulevard, P.O.Box 999, Richland, WA 99352, USA
[2]Center for High Energy Physics, 1274 University of Oregon, Eugene, OR 97403, USA

Abstract. Measurements in Liquid Argon Time Projection Chamber neutrino detectors feature large, high fidelity event images. Deep learning techniques have been extremely successful in classification tasks of photographs, but their application to these event images is challenging, due to the large size of the events, more two orders of magnitude larger than images found in classical challenges like MNIST or ImageNet. This leads to extremely long training cycles, which slow down the exploration of new network architectures and hyperparameter scans to improve the classification performance. We present studies of scaling an LArTPC classification problem on multiple architectures, spanning multiple nodes. The studies are carried out in simulated events in the MicroBooNE detector.

1 Introduction

The MicroBooNE detector is a Liquid Argon Time Projection Chamber (LArTPC) at the Fermi National Accelerator Laboratory (Fermilab). Its technical specifications are described in detail elsewhere [1]. We focus in our report on the data that is recorded by the 8256 wires that are arranged in three readout planes at -60, 60 and 90 degrees with respect to the neutrino beam, with 2400 wires in each of the two induction planes (U and V) and 3456 wires in the collection plane (Y). An interaction of a neutrino from the Fermilab beam [2] with an Argon nucleus results in the production of charge depositions and photons from the ionization and scintillation processes. Tracks and showers result from the charge that drifts to the readout planes, where it is digitized. One digitization waveform results from samples of arriving charge corresponding to a drift distance of 80 μm; one event in the MicroBooNE experiment corresponds to 9600 digitization steps around the interaction time, which is 6.4 ms (3.2 ms) before (after) the beam spill. One uncompressed event corresponds to $\approx$ 150 MB of data. The purpose of this note is to describe a tool that allows the use of multiple GPUs to train deep learning models to classify simulated events in the MicroBooNE detector, and to evaluate its performance. It is structured as follows: Section 2 describes the samples that were used in this study, Section 3 gives a brief description of the network used to categorize the samples, Section 4 gives an overview of the MaTEx tool that underpins our scaling studies, and Section 6 describes our measurements in detail. Finally, we summarize our findings in Section 7.

*e-mail: jan.strube@pnnl.gov

2 Data Sample

The data used in this study was produced in the LArSoft framework. It consists of 10k events for each of the following particles: $e^{\pm}$, K^{-}, μ^{-}, γ, π^{-}.

To simplify the training, we group electrons and positrons into the same category. After simulating the events in `LArSoft` [3], we use our own "KevLAr" framework to validate the output and to convert the files to the HDF5 data format. During this process, we also strip each event of data in the top 1/3 and the bottom 1/3 of time ticks, since the simulated event always occurrs in the central 1/3 of the recorded time ticks. This process results in six files with a total size of about 91 GB. This data is then re-labeled to group the electrons and positrons, shuffled to improve the training performance, and compressed using the `gzip` library with the compression setting of 9 (the highest level). The resulting data set was split into one training file of 30k events, one validation file of 10k events and one testing file of 5k events. In compressed form, these three files take up about 5 GB.

3 Network Performance

The scope of this study is limited to the scaling behavior of the MaTEx implementation of the networks, rather than its physics performance. Nevertheless, we have evaluated the particle separation power of the baseline implementation. The computation of a scoring matrix for different particle species gives us a more direct performance criterion, and one that is more relevant experimentally than the value of the loss function, which averages over all five particle species.

The network we have implemented has a fairly simple convolutional architecture, shown in Figure 1, with 5 blocks of alternating convolutional and pooling layers, followed by a densely connected hidden layer before the output layer. We have chosen exponential linear units (ELU) [4] as activation functions to help alleviate the vanishing gradient problem with our extremely sparse data. The ELU is defined according to Equation 1. The non-zero values of the function for $x < 0$ effectively avoid neuron death that we encountered with other choices for activation functions.

$$f(x) = \begin{cases} \alpha(e^{x} - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{1}$$

Table 1. Output of the classification network. The weights in each row add up to the number of input events for this particle type. The score column indicates how many times the highest score was assigned to the correct category.

	Truth	Prediction (sum of weights)					
input		γ	$e^{\pm}$	μ^{-}	π^{-}	K^{-}	score
873	γ	851.47	21.53	0.00	0.00	0.00	852
1622	$e^{\pm}$	8.19	1611.79	0.00	0.01	2.00	1613
856	μ^{-}	0.00	0.00	853.93	0.00	2.07	854
826	π^{-}	2.90	3.87	3.00	483.68	332.54	482
823	K^{-}	1.00	1.00	9.43	307.19	504.37	508

4 Brief Introduction to `MaTEx`

The MaTEx toolkit [5, 6] allows the training of deep neural networks using multiple GPU nodes. To achieve this, data is split across the different nodes, such that each node sees

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 3600, 3600, 10)    260
_________________________________________________________________
elu_1 (ELU)                  (None, 3600, 3600, 10)    0
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 720, 720, 10)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 720, 720, 64)      16064
_________________________________________________________________
elu_2 (ELU)                  (None, 720, 720, 64)      0
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 144, 144, 64)      0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 144, 144, 128)     204928
_________________________________________________________________
elu_3 (ELU)                  (None, 144, 144, 128)     0
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 128)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 256)       819456
_________________________________________________________________
elu_4 (ELU)                  (None, 28, 28, 256)       0
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 5, 5, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
fc1 (Dense)                  (None, 32)                204832
_________________________________________________________________
elu_5 (ELU)                  (None, 32)                0
_________________________________________________________________
predictions (Dense)          (None, 5)                 165
=================================================================
Total params: 1,245,705
Trainable params: 1,245,705
Non-trainable params: 0
```

Figure 1. Simplified description of the network layers including the number of parameters in each layer

the same data throughout the training. Weight updates are propagated to the nodes using MPI and gradients from all nodes are averaged before computing the updated weights [7]. All of this is transparent to the user, such that the only change to the user workflow are the loading of `MaTEx` instead of vanilla tensorflow, and the change to use the `MaTEx` dataset

instead of the tensorflow dataset. The source code is available under an open source license at https://github.com/matex-org/matex.

5 Data Preparation

The training data is too large to fit into the memory of one of the available nodes. On the other hand, disk I/O is too slow to achieve reasonable training rates when reading data from disk repeatedly. We are therefore storing the data in compressed form, using the blosc library, which in turn employs the lz4 compression. This required some modifications to the MaTEx dataset object. Instead of using a numpy array to store the data, each compressed event is stored in a dictionary, with the original index as the dictionary key. This allows using the same `data[index]` syntax in the code, but batches can no longer be retrieved as slices of the data array. Instead, for batch sizes other than 1, we allocate an empty array of the same size as the batch, and then uncompress the events for the batch one-by-one and copy them into the empty array.

6 Measurements

To determine the scaling properties of our setup across multiple nodes, we used the resources provided by the PNNL Institutional Computing (PIC) infrastructure. The site has up to 20 nodes available, each equipped with 2 NVidia P-100 GPUs. For each experiment, we reserve an integral number of nodes to be independent of other possible loads. Our time measurements consist of simple calls to the Python `time.time()` function. The measurements in this section are predominantly to study the computational performance of our setup. The evaluation and improvement of the physics performance of different network architectures is left for future studies.

6.1 Data Loading

The `MaTEx` data set automatically splits itself across all available nodes, such that each node contains a different 1/N fraction of the data. We test the scaling behavior of the data access patterns by running an empty network that simply returns unity regardless of the input data. We run this job across different numbers of nodes. The baseline job is run on 1 GPU, but we reserve the whole node, so the second GPU is idle. The other experiments are run on different numbers of nodes, where they use both GPUs each time. The results are shown in Figure 2. The `HDF5` library was not compiled with MPI, so data throughput does not scale with the number of nodes. Nevertheless, it is obvious that even in the default setting, the library scales extremely well with the number of nodes. In addition to being able to distribute the work of loading the data from disk, uncompressing it and re-packaging it into the `MaTEx` data set using `blosc`, experiments using larger number of nodes require less memory on each node, resulting in fewer allocations. We believe that this explains at least partly the observed speedup for larger number of nodes. Other factors could be related to disk access patterns of unrelated processes on the shared file system. While many of these effects are not a direct result of using MaTEx, they are relevant aspects of measurements under realistic conditions and demonstrate the multi-fold benefits of distributing large workloads across multiple workers.

6.2 Training Performance

Since the confusion matrix in Table 1 does not provide a straightforward way to evaluate the performance, we use categorical cross entropy to evaluate the performance for each experiment. It is defined as $H(p,q) = -N \sum_x p(x) \log(q(x))$, where $p(x)$ is the true distribution of

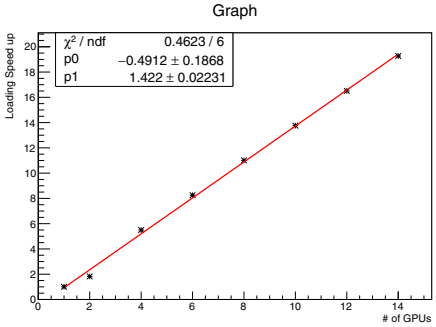

Figure 2. Speedup of the data loading on different numbers of nodes.

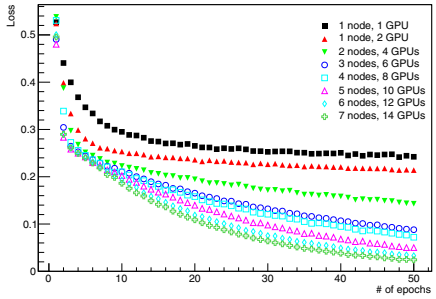

Figure 3. Value of the loss function on training data after different numbers of epochs. Different numbers of nodes are represented by different symbols.

categories in the event, and $q(x)$ is the predicted distribution. In this way, the training assigns equal weight to all categories.

6.2.1 Performance across different numbers of GPUs

We found that our data and network architecture quickly saturate the available memory on a GPU. The following studies are carried out with a batch size of 2 events per GPU. Figure 3 shows the scaling behavior of the network across different numbers of GPUs. For larger number of nodes, the network arrives at a better performance *faster* than for a comparable number of epochs using fewer nodes. For example, the value of the loss function for single-node training after 10 epochs can be achieved with 14 GPUs after only 2 epochs, resulting effectively in a 50-fold speed up. Additionally, single node-training flattens out at much larger values of the loss function, indicating a less effective training. This behavior can be explained when remembering that `MaTEx` averages gradients from all workers. Effectively, the batch size scales with the number of workers. The batch size is generally considered a hyper parameter, whose optimal value can be determined in a parameter scan. For our experiment, we expect the optimal value to be much larger than our available number of nodes, such that more nodes (or larger batches) lead to more effective gradient updates and hence more optimal training. In this way, we have demonstrated the strong scaling performance of our `MaTEx` setup. In addition to this effect of achieving better network performance for training on a larger num-

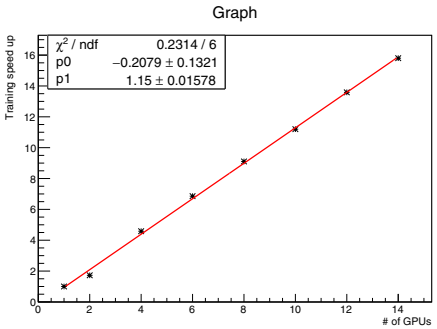

Figure 4. Scaling behavior of the computational performance of training 50 epochs, normalized to the performance on one GPU.

ber of GPUs, we have also measured the time to train the same amount of epochs across different numbers of GPUs, resulting from distributing the data across multiple nodes. For these tests we are not including the validation step after each epoch, which incurs additional traffic between nodes and could potentially change the scaling behavior. Figure 4 shows that the time for training a certain number of epochs scales excellently with the number of GPUs. The deviation from the expected line of slope one in the scaling behavior is most likely due to the in-memory de-compression of our data before it gets moved to the GPU. Additionally, there might be some residual I/O effects that are alleviated by increasing the number of nodes. We have confirmed that these measurements are repeatable, but the detailed understanding of the exact scaling behavior would require investigating many effects, including memory access patterns and inter-node communication protocols. Such investigations are outside of the scope of this note.

6.2.2 Performance across different numbers of CPUs

We have also measured the scaling performance of training on CPUs. While the training on a single GPU is significantly faster than on a CPU, it is interesting to evaluate the scaling performance on different architectures, since most HPC systems are hybrid systems, and on some systems CPU cycles might be more readily availability than GPU. Our tests were carried out on Dual Intel Broadwell E5-2620 v4 CPUs with 16 cores and 64 GB 2133 MHz DDR4 memory per node. Due to the much slower processing on CPU, we only trained our networks for 10 epochs instead of 50. The training on 1 and 2 nodes did not complete within the time limits for our available queues. The scaling behavior on different numbers of nodes is shown in Figure 5. The slope in this case is consistent with linear scaling, indicating that the I/O effect of GPU training is less pronounced in this study.

7 Summary

Events in liquid argon TPC detectors can be represented as large multi-color images, and convolutional neural networks can be effective in identifying and categorizing events in these detectors. We have demonstrated a simple convolutional architecture that is able to categorize simulated single particles in the MicroBooNE detector. The size of the images renders training methods on single nodes ineffective due to the excessively large turnaround time and worse training pattern. Using open source software for in-memory compression and the the

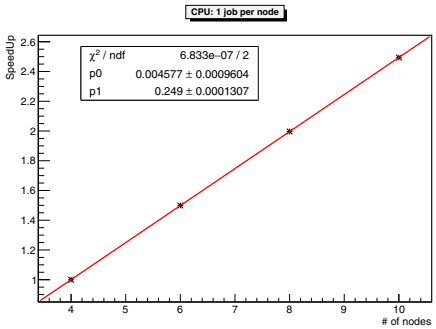

Figure 5. Scaling behavior of the computational performance of training 10 epochs, normalized to the performance of four CPUs

MaTEx toolkit, we have demonstrated excellent scaling behavior of the training on as many as 14 GPUs distributed across seven nodes. In addition to the weak scaling that is achieved by distributing the computations across multiple workers, we have demonstrated a strong scaling behavior that results from the benefits of an effectively larger batch size on more workers. We find that the training scales linearly with the number of CPUs, and we observe a relative speedup of around 35 of GPU vs. CPU training. This speedup is much smaller than that observed for training on smaller images and indicates that data movement takes up a significant fraction of the training time. Future computational improvements should therefore attempt to focus on reducing the data movement first.

Tuning the performance of deep neural networks frequently involves the tuning of hyper parameters, which requires training many iterations over multiple epochs each. Such studies are orthogonal to the studies presented in this note and are left for future endeavors.

8 Acknowledgments

The research was performed using PNNL Institutional Computing at Pacific Northwest National Laboratory.

References

[1] R. Acciarri, C. Adams, R. An, A. Aparicio, S. Aponte, J. Asaadi, M. Auger, N. Ayoub, L. Bagby, B. Baller et al., Journal of Instrumentation **12**, P02017 (2017)
[2] (2018), http://www.hep.princeton.edu/mumu/target/Numi/moore_150813.pdf
[3] E.D. Church, arXiv e-prints (2013), 1311.6774
[4] D.A. Clevert, T. Unterthiner, S. Hochreiter, arXiv e-prints (2015), 1511.07289
[5] A. Vishnu, C. Siegel, J. Daily, arXiv e-prints arXiv:1603.02339 (2016), 1603.02339
[6] A. Vishnu, J. Manzano, C. Siegel, J. Daily, arXiv e-prints arXiv:1704.04560 (2017), 1704.04560
[7] V. Amatya, A. Vishnu, C. Siegel, J. Daily, arXiv e-prints arXiv:1709.03316 (2017), 1709.03316