**Issues in the Reproducibility of Deep Learning Results**

*S. Jean-Paul, T. Elseify, I. Obeid and J. Picone*

The Neural Engineering Data Consortium, Temple University, Philadelphia, Pennsylvania, USA

{tuh26880, tug98850, iobeid, picone}@temple.edu

The Neuronix high-performance computing cluster allows us to conduct extensive machine learning experiments on big data [1]. This heterogeneous cluster uses innovative scheduling technology, Slurm [2], that manages a network of CPUs and graphics processing units (GPUs). The GPU farm consists of a variety of processors ranging from low-end consumer grade devices such as the Nvidia GTX 970 to higher-end devices such as the GeForce RTX 2080. These GPUs are essential to our research since they allow extremely compute-intensive deep learning tasks to be executed on massive data resources such as the TUH EEG Corpus [2]. We use TensorFlow [3] as the core machine learning library for our deep learning systems, and routinely employ multiple GPUs to accelerate the training process.

Reproducible results are essential to machine learning research. Reproducibility in this context means the ability to replicate an existing experiment – performance metrics such as error rates should be identical and floating-point calculations should match closely. Three examples of ways we typically expect an experiment to be replicable are: (1) The same job run on the same processor should produce the same results each time it is run. (2) A job run on a CPU and GPU should produce identical results. (3) A job should produce comparable results if the data is presented in a different order. System optimization requires an ability to directly compare error rates for algorithms evaluated under comparable operating conditions. However, it is a difficult task to exactly reproduce the results for large, complex deep learning systems that often require more than a trillion calculations per experiment [5]. This is a fairly well-known issue and one we will explore in this poster.

Researchers must be able to replicate results on a specific data set to establish the integrity of an implementation. They can then use that implementation as a baseline for comparison purposes. A lack of reproducibility makes it very difficult to debug algorithms and validate changes to the system. Equally important, since many results in deep learning research are dependent on the order in which the system is exposed to the data, the specific processors used, and even the order in which those processors are accessed, it becomes a challenging problem to compare two algorithms since each system must be individually optimized for a specific data set or processor. This is extremely time-consuming for algorithm research in which a single run often taxes a computing environment to its limits. Well-known techniques such as cross-validation [5,6] can be used to mitigate these effects, but this is also computationally expensive.

These issues are further compounded by the fact that most deep learning algorithms are susceptible to the way computational noise propagates through the system. GPUs are particularly notorious for this because, in a clustered environment, it becomes more difficult to control which processors are used at various points in time. Another equally frustrating issue is that upgrades to the deep learning package, such as the transition from TensorFlow v1.9 to v1.13, can also result in large fluctuations in error rates when re-running the same experiment. Since TensorFlow is constantly updating functions to support GPU use, maintaining an historical archive of experimental results that can be used to calibrate algorithm research is quite a challenge. This makes it very difficult to optimize the system or select the best configurations.

1. Research reported in this publication was most recently supported by the National Science Foundation Partnership for Innovation award number IIP-1827565. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
2. Research reported in this publication was also supported by the National Human Genome Research Institute of the National Institutes of Health under award number U01HG008468. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

The overall impact of all of these issues described above is significant as error rates can fluctuate by as much as 25% due to these types of computational issues. Cross-validation is one technique used to mitigate this, but that is expensive since you need to do multiple runs over the data, which further taxes a computing infrastructure already running at max capacity.

GPUs are preferred when training a large network since these systems train at least two orders of magnitude faster than CPUs [7]. Large-scale experiments are simply not feasible without using GPUs. However, there is a tradeoff to gain this performance. Since all our GPUs use the NVIDIA CUDA® Deep Neural Network library (cuDNN) [8], a GPU-accelerated library of primitives for deep neural networks, it adds an element of randomness into the experiment. When a GPU is used to train a network in TensorFlow, it automatically searches for a cuDNN implementation. NVIDIA’s cuDNN implementation provides algorithms that increase the performance and help the model train quicker, but they are non-deterministic algorithms [9,10]. Since our networks have many complex layers, there is no easy way to avoid this randomness. Instead of comparing each epoch, we compare the average performance of the experiment because it gives us a hint of how our model is performing per experiment, and if the changes we make are efficient.

In this poster, we will discuss a variety of issues related to reproducibility and introduce ways we mitigate these effects. For example, TensorFlow uses a random number generator (RNG) which is not seeded by default. TensorFlow determines the initialization point and how certain functions execute using the RNG. The solution for this is seeding all the necessary components before training the model. This forces TensorFlow to use the same initialization point and sets how certain layers work (e.g., dropout layers). However, seeding all the RNGs will not guarantee a controlled experiment. Other variables can affect the outcome of the experiment such as training using GPUs, allowing multi-threading on CPUs, using certain layers, etc.

To mitigate our problems with reproducibility, we first make sure that the data is processed in the same order during training. Therefore, we save the data from the last experiment and to make sure the newer experiment follows the same order. If we allow the data to be shuffled, it can affect the performance due to how the model was exposed to the data. We also specify the float data type to be 32-bit since Python defaults to 64-bit. We try to avoid using 64-bit precision because the numbers produced by a GPU can vary significantly depending on the GPU architecture [11-13]. Controlling precision somewhat reduces differences due to computational noise even though technically it increases the amount of computational noise.

We are currently developing more advanced techniques for preserving the efficiency of our training process while also maintaining the ability to reproduce models. In our poster presentation we will demonstrate these issues using some novel visualization tools, present several examples of the extent to which these issues influence research results on electroencephalography (EEG) and digital pathology experiments and introduce new ways to manage such computational issues.

References

1. C. Campbell, N. Mecca, I. Obeid, and J. Picone, “The Neuronix HPC Cluster: Improving Cluster Management Using Free and Open Source Software Tools,” *IEEE Signal Processing in Medicine and Biology Symposium*, 2017, p. 1.
2. A. B. Yoo, M. A. Jette, and M. Grondona, “SLURM: Simple Linux Utility for Resource Management,” in *Job Scheduling Strategies for Parallel Processing*, 2003, pp. 44–60.
3. I. Obeid and J. Picone, “The Temple University Hospital EEG Data Corpus,” in *Augmentation of Brain Function: Facts, Fiction and Controversy. Volume I: Brain-Machine Interfaces*, 1st ed., vol. 10, Lausanne, Switzerland: Frontiers Media S.A., 2016, pp. 394–398.
4. M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” *arXiv: 1603.04467*, p. 19, Jan. 2015. ﻿https://arxiv.org/pdf/1603.04467.pdf.
5. J. Gardner, Y. Yang, R. Baker, and C. Brooks, “Enabling End-To-End Machine Learning Replicability: A Case Study in Educational Data Mining,” *arXiv:1806.05208v2 [cs.LG]*, p. 10, Jun. 2018.
6. R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York City, New York, USA: John Wiley & Sons, Inc, 2001.
7. Google, “Using GPUs for training models in the cloud,” 2019. [Online]. Available: https://cloud.google.com/ml-engine/docs/using-gpus. [Accessed: 31-Oct-2019].
8. S. Chetlur *et al.*, “cuDNN: Efficient Primitives for Deep Learning,” *arXiv: 1410.0759*, p. 9, 2014.
9. P. Nagarajan, G. Warnell, and P. Stone, “The Impact of Nondeterminism on Reproducibility in Deep Reinforcement Learning,” in *2nd Reproducibility in Machine Learning Workshop*, 2018, p. 10.
10. P. Nagarajan, G. Warnell, and P. Stone, “Deterministic Implementations for Reproducibility in Deep Reinforcement Learning,” in *AAAI 2019 Workshop on Reproducible AI*, 2019, p. 17.
11. M. Taufer, O. Padron, P. Saponaro, and S. Patel, “Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs,” in *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010, pp. 1–9.
12. D. Yablonski, “Numerical Accuracy Differences in CPU and GPGPU Codes,” Northeastern University, 2011.
13. D. Fay, A. Sazegari, and D. Connors, “A Detailed Study of the Numerical Accuracy of GPU-Implemented Math Functions,” in *Proceedings of the Supercomputing Workshop on General Purpose GPU Computing: Practice and Experience*, 2006, p. 1.