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.

NEURAL ENGINEERING DATA CONSORTIUM
www.nedcdata.org

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

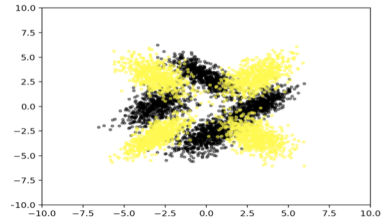
College of Engineering
Temple University

Abstract

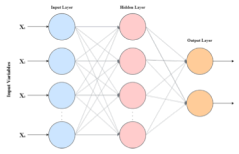
- **Reproducible result**: a researcher should be able to generate performance comparable to a published baseline using the same data and algorithm but a different computer architecture.
- Reproducible results are essential to machine learning research since system optimization requires an ability to directly compare error rates for algorithms evaluated under comparable conditions.
- GPUs are preferred when training a large network since these systems train at least an order of magnitude faster than CPUs.
- GPUs, however, produce different numeric results than CPUs. The degree of difference is, of course, architecture specific, compounding the problem.
- Most deep learning toolboxes use a math library known as CUDA which also introduce randomness. State of the art systems typically use multiple GPUs for training, which adds even more randomness.
- In this study, two popular machine learning libraries, PyTorch and TensorFlow, were analyzed.
- Seeding techniques specific to individual architectures are used to mitigate the effects of non-deterministic behavior.

Introduction

- There were a total of three unique datasets used for training and evaluation purposes:
  - ❑ 2D (10K train, 2K dev, 2K eval): two-dimensional randomly generated multivariate Gaussian data:



  - ❑ 5D (100K train, 10K dev, 10K eval): five-dimensional multivariate Gaussian data with covariance matrices that resemble EEG data.
  - ❑ 26D (18,936 train, 1,154 eval): 26-dimensional EEG vectors extracted from seizure events in the TUH EEG Seizure Detection Corpus.
- The data sets are publicly available at:
  www.isip.piconepress.com/courses/temple/ece_8527/resources/data/
- An MLP model with 26 hidden nodes, Adam optimization and a categorical entropy loss function was used for all experiments. The software is also available at the above URL.

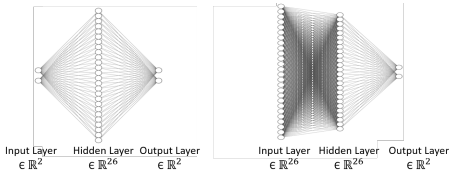


Mismatched Architectures

- Four unique devices used:
  - ❑ CPU: Intel Xeon(R) CPU E5-2620 v4 @ 2.10GHz
  - ❑ GTX 1070: 8GB GDDR5, 256 GB/s Bandwidth
  - ❑ Tesla P40: 24GB GDDR5X, 480 GB/s Bandwidth
  - ❑ RTX 2080 : 8GB GDDR6, 448 GB/s Bandwidth
- Software packages used with Python 3:
  - ❑ PyTorch v1.3.1
  - ❑ TensorFlow v1.14 / Keras v2.3.1
  - ❑ CUDA v10.1.243
- Cross-platform evaluations:

Processor		2D		5D		Set 04	
Train	Eval	PyTorch	TensorFlow	PyTorch	TensorFlow	PyTorch	TensorFlow
CPU	CPU	8.35%	8.80%	36.93%	36.85%	48.23%	48.83%
CPU	RTX 2080	8.35%	8.80%	36.93%	36.85%	48.23%	48.83%
CPU	Tesla P40	8.35%	8.80%	36.93%	36.85%	48.23%	48.83%
CPU	GTX 1070	8.35%	8.80%	36.93%	36.85%	48.23%	48.83%
RTX 2080	CPU	8.60%	8.85%	36.79%	36.90%	49.18%	48.92%
RTX 2080	RTX 2080	8.60%	8.85%	36.79%	36.90%	49.18%	48.92%
RTX 2080	Tesla P40	8.60%	8.85%	36.79%	36.90%	49.18%	48.92%
RTX 2080	GTX 1070	8.60%	8.85%	36.79%	36.90%	49.18%	48.92%
Tesla P40	CPU	8.65%	8.80%	36.77%	36.64%	48.83%	48.74%
Tesla P40	RTX 2080	8.65%	8.80%	36.77%	36.64%	48.83%	48.74%
Tesla P40	Tesla P40	8.65%	8.80%	36.77%	36.64%	48.83%	48.74%
Tesla P40	GTX 1070	8.65%	8.80%	36.77%	36.64%	48.83%	48.74%
GTX 1070	CPU	8.85%	8.80%	36.96%	36.69%	49.52%	48.83%
GTX 1070	RTX 2080	8.85%	8.80%	36.96%	36.69%	49.52%	48.83%
GTX 1070	Tesla P40	8.85%	8.80%	36.96%	36.69%	49.52%	48.83%
GTX 1070	GTX 1070	8.85%	8.80%	36.96%	36.69%	49.52%	48.83%
	Average	8.61%	8.81%	36.86%	36.77%	48.94%	48.83%
	StDev	0.18%	0.02%	0.09%	0.11%	0.50%	0.06%

- Statistical Significance:
  - ❑ 2D: +/- 1.40% [7.21%, 10.01%]
  - ❑ 5D: +/- 1.12% [35.74%, 37.98%]
  - ❑ 26D: +/- 3.42% [45.52%, 52.36%]
- The t-tests performed on the three sets suggest as the feature's dimensionality increases, the difference in results between TensorFlow and PyTorch becomes less significant.
- The weights of the model were initialized randomly using a constant seeding value.
- Model parameters (e.g., weights) were not reproducible in TensorFlow.
- The error rates were reproducible when the computing architecture remained constant.



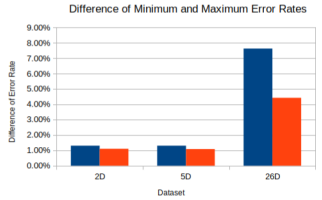
- A network containing a single 26 neuron hidden layer with lower dimensional input (i.e. 2D) tend to learn redundant information.
- Since the number of nodes in the hidden layer remained constant, inputs with a higher feature dimension increased the number of weights in the model in the input layer.
- TensorFlow and PyTorch invoke reduction functions that introduce non-deterministic behavior on GPU and multi-threaded CPU architectures.
- This non-determinism is caused by threads not performing operations in the same order after each run. It is impossible to control these things from high-level interfaces (e.g., Keras).

Statistical Analysis

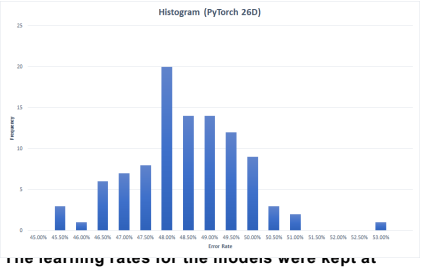
- To determine the effect of varying initial conditions, a different model was initialized with 100 unique large prime numbers as the seeds.

	Error Rate											
	Set_07: 2D				Set_07: 5D				Set_04: 26D			
System	Min	Max	Average	StDev	Min	Max	Average	StDev	Min	Max	Average	StDev
PyTorch	8.0500%	9.3500%	8.5250%	0.2796%	36.2600%	37.6600%	36.8086%	0.2521%	45.0216%	52.6407%	48.1923%	1.3698%
TensorFlow	8.5000%	9.6000%	8.8940%	0.2015%	36.3200%	37.4000%	36.7918%	0.2266%	44.8485%	49.2641%	47.2788%	0.9820%

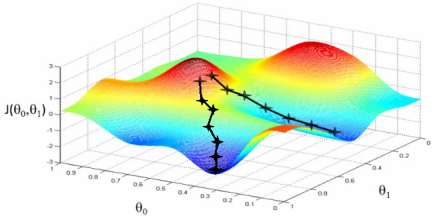
- The results showed that the initial starting point of the model's weights has a sizeable effect on its performance. Depending on the seed used, a model with the same architecture produced error rates with a difference of over 7%.



- The input feature dimensionality was a determining factor for how much the error rates varied. The 2 and 5-dimensional datasets produced error rates with a significantly smaller standard deviation than the 26-dimensional dataset.
- PyTorch consistently produced error rates with a higher standard deviation than TensorFlow for all datasets that were evaluated.

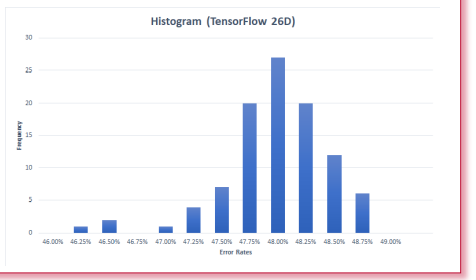


- The learning rates for the models were kept at 0.0005. This prevented the models from diverging, but also reduced their potential to explore the error space.
- The distinct random starting points caused the models to converge at different local minima resulting in different error rates. This is a well-known problem in gradient descent approaches.



Sensitivity to Data Ordering

- The ordering of the data influences the performance of a model due to how the weights become initialized. The effect of this was measured by creating a single model and fitting a shuffled version of the data after each run.
- There were a total of 100 runs where the training data was shuffled with a unique seed each run.
- Despite keeping the model architecture constant, changing the order of the data resulted in error rates within 3% absolute of one another.
- Ensuring the same ordering of data can help prevent variations in performance, but this is application specific.



Summary

- It is difficult to achieve reproducibility when using popular deep learning software packages such as PyTorch and TensorFlow. Their dependence on CUDA functions compounds the problem. Parallelization and floating-point precision (e.g., CPU vs. GPU) play a significant role also.
- Seeding all the random number generators helps mitigate the error in performance by ensuring the model is initialized with the same random weights.
- The complexity of the model influences the variance in error rates.
- The effects of increased model and data complexity on reproducibility will be a topic explored in future experiments.

Acknowledgements

- Research reported in this publication was most recently 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.
- This material is also based in part upon work supported by the National Science Foundation under Grant No. CNS-1726188. 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.