Accelerating Real-Time Imaging for Radiotherapy: Leveraging Multi-GPU Training with PyTorch

Ruth Obe
Dept. of Computer Science and of Software Engineering
U. of Houston—Clear Lake, USA

Brandt Kaufmann
Dept. of Mathematics and Statistics
U. of San Francisco, USA

Kaelen Baird
Dept. of Computer Science and of Mathematics
Skidmore College, USA

Sam Kadel
Dept. of Computer Science and of Psychology
Mount Holyoke College, USA

Yasmin Soltani
Dept. of Biomedical Engineering
U. of Houston, USA

Mostafa Cham
Dept. of Information Systems
U. of Maryland, Baltimore County

Matthias K. Gobbert
Dept. of Mathematics and Statistics
U. of Maryland, Baltimore County

Carlos A. Barajas
Dept. of Mathematics and Statistics
U. of Maryland, Baltimore County

Zhuoran Jiang
Medical Physics Graduate Program
Duke University, USA

Vijay R. Sharma
Dept. of Radiation Oncology
U. of Maryland School of Medicine, USA

Lei Ren
Department of Radiation Oncology
U. of Maryland School of Medicine, USA

Stephen W. Peterson
Dept. of Physics
U. of Cape Town, South Africa

Jerimy C. Polf
H3D, Inc.
USA

***Abstract*—Proton beam therapy is an advanced form of cancer radiotherapy that uses high-energy proton beams to deliver precise and targeted radiation to tumors. This helps to mitigate unnecessary radiation exposure in healthy tissues. Real-time imaging of prompt gamma rays with Compton cameras has been suggested to improve therapy efficacy. However, the camera's non-zero time resolution leads to incorrect interaction classifications and noisy images that are insufficient for accurately assessing proton delivery in patients. To address the challenges posed by the Compton camera's image quality, machine learning techniques are employed to classify and refine the generated data. These machine-learning techniques include recurrent and feedforward neural networks. A PyTorch model was designed to improve the data captured by the Compton camera. This decision was driven by PyTorch's flexibility, powerful capabilities in handling sequential data, and enhanced GPU usage. This accelerates the model's computations on large-scale radiotherapy data. Through hyperparameter tuning, the validation accuracy of our PyTorch model has been improved from an initial 7% to over 60%. Moreover, the PyTorch Distributed Data Parallelism strategy was used to train the RNN models on multiple GPUs, which significantly reduced the training time with a minor impact on model accuracy.**

***Index Terms*—Proton beam therapy, Compton camera, Classification, Recurrent neural network, PyTorch, Distributed Data Parallelism.**

I. Introduction

Proton beam therapy is an advanced form of cancer radiotherapy that uses high-energy proton beams to deliver precise and targeted radiation to tumors mitigating unnecessary radiation exposure [14]. Unlike x-ray therapies, which go through the entire body, proton beams release the majority of their energy in a more localized area. This localized release of energy is the *Bragg peak*, which allows more precise radiation delivery [14]. Since proton beam therapy applies more radiation in a smaller radius, it is especially important to know where the beam is in relation to tumors. In order to take full advantage of proton beam therapy and prevent damaging healthy tissue when patients move, clinicians need an efficient technique to image prompt gamma rays in real time.

Utilizing real-time imaging of prompt gamma rays can enhance the effectiveness of this therapy. Compton cameras are proposed for this purpose, capturing prompt gamma rays emitted by proton beams as they traverse a patient's body [3], [13]. However, the Compton camera's non-zero time resolution results in the simultaneous recording of interactions, causing reconstructed images to be noisy and lacking the necessary level of detail to assess proton delivery for the patient effectively. The noise in the Compton camera causes uncertainty about the location of the proton beam when radiating tissues which can cause healthy tissue to be radiated possibly leading to future complications [3], [7], [8], [13].

To address the challenges posed by the Compton camera's resolution and its impact on image quality, machine learning techniques, such as recurrent and deep neural networks, are employed to classify the prompt gamma event ordering to improve the clarity of the proton beam during treatment [8].

These trained models clean the raw Compton camera data by identifying and removing false data before image reconstruction. These advanced learning algorithms can effectively distinguish various interaction types and enhance the captured information while reducing external noise in the imaging. This type of real-time imaging leads to more precise evaluations of the radiation delivery during the patient's treatment. It ensures the whole tumor gets the appropriate radiation levels for successful treatment.

We designed feed-forward neural networks (FNN) and recurent neural networks (RNN) models in PyTorch to enhance data captured by the Compton camera. The decision to develop these models with the PyTorch library over the commonly used TensorFlow library [3], [7], [8], [13] was driven by PyTorch's flexibility, powerful capabilities in handling sequential data, enhanced GPU usage, and simple Python-like syntax, accelerating the model's computations and further optimizing the processing of large-scale data and allowing for faster development and training of new model types [10]. The model successfully demonstrated speedier training performance than previous approaches and achieved fair accuracy with limited hyperparameter tuning. It highlights its effectiveness in advancing real-time imaging of prompt gamma rays for enhanced evaluation of proton delivery in cancer therapy.

The remainder of this paper is organized as follows: Section II covers proton beam radiation and the background information on imaging, prompt-gamma interactions, and machine learning models. Section III covers our translation of the models to PyTorch, configuration of distributed data parallel (DDP) training, and hardware and software used for model training. Section IV covers the results of our hyperparameter study on our deep and recurrent neural networks and multi-GPU training results. Section V wraps up our research by concluding our findings.

II. Application Background

A. Proton Beam Therapy

Radiation therapy is an effective approach for treating cancer by utilizing powerful radiation to eradicate cancer cells. X-ray therapy is frequently utilized in cancer treatment. However, a large portion of the radiation is delivered as it enters the body. Unfortunately, radiation therapy often fails to provide a sufficiently concentrated dose to the tumor while providing a similar dose to healthy tissue. Moreover, x-rays pass through the entire body, causing unavoidable radiation exposure. In contrast, proton therapy, another type of radiation treatment, offers enhanced efficiency in addressing these issues [14].

In contrast to x-ray therapy, proton therapy concentrates most of the radiation dosage at the tumor site instead of the point of entry. This precise of targeting significantly improves the treatment's effectiveness. Moreover, proton therapy outperforms x-ray therapy by limiting the penetration of the proton beam to the tumor area, thereby reducing the exposure of surrounding tissues [3], [8], [9], [13].

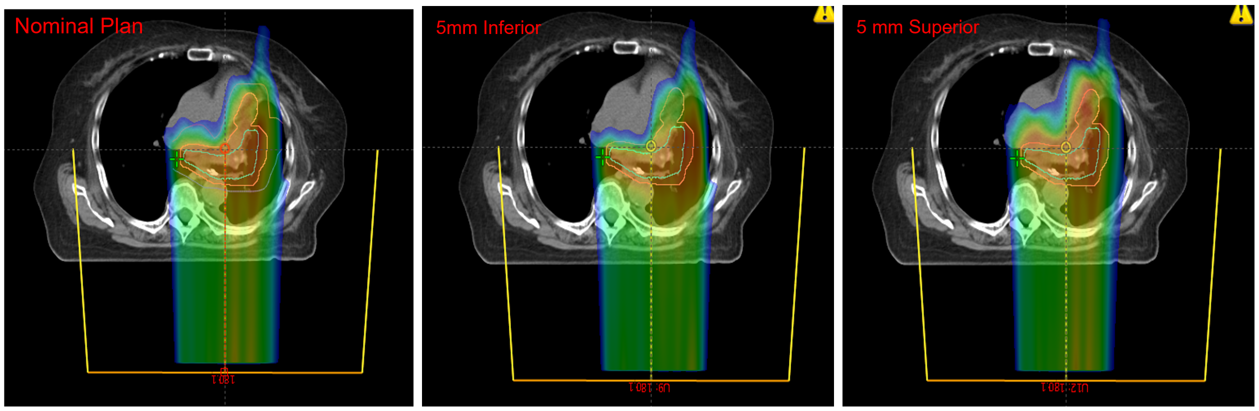
The tumor size determines whether a step-by-step elimination of tumor cells using the radiation beam is required. To guarantee that every part of the tumor receives the necessary radiation dosage, healthcare professionals create a safety margin. This margin expands the treatment area and accounts for the patient's slight movements or positional variations during several weeks of treatment, to guarantee that all of the tumor is treated [14]. But this safety margin may encroach on healthy tissue that should not receive treatment. Figure 1 visualizes the issue. Figure 1 (a) shows the desired outcome of the treatment by a proton beam from the bottom of the scan. Figure 1 (b) illustrates the situation of the patient moving slightly upward during the treatment, cause an undershoot of the proton beam from the bottom of the scan. While Figure 1 (c) illustrates an overshoot because of the patient moving upward during the treatment.

Having real-time information about the path of the proton beam inside the patient's body during treatment could reduce the safety margin's size and protect healthy tissue better. One proposed solution for obtaining such information is using a Compton camera, which can capture immediate images of prompt gamma rays emitted by the proton beams as they pass through the body. These cameras offer valuable insights into the beam's location, facilitating more precise and efficient treatment.

B. Compton Camera and Image Reconstruction

Compton cameras are advanced multistage detectors used to image proton beams used in proton beam therapy. When protons pass through the human body, they interact with atoms, resulting in the emission of prompt gamma rays. As these gamma rays exit the body, some of them collide with the modules in the Compton camera [14]. The camera's modules then measure the energy and position of the prompt gamma rays as they traverse different detection stages. Each recorded Compton scatter includes x-, y-, and z-coordinates, as well as the corresponding energy level. These recorded interactions, known as *events*, provide raw output data in the form (e_i, x_i, y_i, z_i), where $i = 1, 2, 3$, and e_i represents the energy level [1], [7], [8], [14].

Sophisticated algorithms exist for reconstructing the path of the proton beam based on the data obtained from the Compton camera. By utilizing the camera's ability to generate complete 3D images of the proton beam's range, it becomes possible to compare the planned treatment dose with the patient's CT scan and make any necessary adjustments. In radiotherapy, it is crucial to ensure conformity between the treatment plan and its execution, ensuring that the patient's bone and soft tissue landmarks are aligned as intended during treatment planning. Even minor movements such as changes in position, fidgeting, scratching, or looking away can disrupt the treatment plan. By obtaining reliable information about the patient's condition from the reconstructed images, clinicians have a better chance of ensuring that the entire tumor receives the precise dose planned while ensuring the safety of surrounding healthy tissues.



(a) normal (b) undershoot (c) overshoot

Fig. 1: Distal range uncertainties of beam therapy.

C. Scatter Types

When the proton beam passes through tissue, prompt gammas are emitted at speeds close to the speed of light. Since all of the interactions in an event happen almost simultaneously, the Compton camera cannot decode the proper ordering of the interaction in an event. This is where scatter types help to identify false events causing noise in the image. There are 13 classes of scatterings, in three groups by type:

1) **True Triple:** The true triple events happen when the camera captures the path of the prompt gamma with a triple interaction in the event. Notice that the ordering of the interactions can be one of six combinations of true triple scattering: 123, 132, 213, 231, 312, 321. As the data is currently implemented, only the 123 ordering of the triple event is usable for image reconstruction [14].
2) **Double-to-Triples (DtoT):** The DtoT events happen when the Compton camera detects an event composed of double and single interaction that are independent of each other [14]. There are six possible ordering of DtoT events: 124, 134, 214, 234, 324, 314. Interaction “4” in the ordering refers to the second prompt gamma interaction in the misdetection events [14].
3) **False Triple:** The false triple events happen when the Compton camera detects a true triple when in reality, the event is composed of three single independent interactions [14].

D. Machine Learning in Image Reconstruction

In our case — and as it often is in medical imaging — our data is extremely noisy, and getting significantly more raw data is not cheap or efficient. This is where machine learning comes in. Machine learning has a rich history in image processing and reconstruction. Since 2006, deep learning has been known to have the capability to recognize target objects in images, and since has been used and improved upon [5]. The data obtained from the Compton cameras are extremely noisy, as there are limitations in the imaging provided, primarily due to their inability to gauge the timing of interactions of gamma particles. Because of the noise, the raw images from the cameras are far from the required accuracy for use in proton beam therapy. They must be thoroughly clean to be precise enough to be used and trusted when using proton beams. This cleaning process uses machine learning to predict the initial proton beams based on the messy result returned by the Compton cameras.

III. Methodology

A. Deep Neural Networks

A neural network with more than three hidden layers is called a Deep Neural Network (DNN). Deep neural networks employ deep learning techniques to generate a suitable representation of the input data required for a specific task. However, one significant challenge posed by the implementation of these deep neural networks is the diminishment of human interpretability. The decision-making processes of these sophisticated methodologies are primarily non-transparent and obscure, thereby creating a ”black box” scenario [5]. Despite the drawback of reduced interpretability in deep learning methodologies, they are crucial when handling vast and high-dimensional datasets. Traditional machine learning techniques can struggle to find meaningful patterns in such complex data due to their limitations in managing multidimensionality and sheer volume. Therefore, the capability of deep learning to model intricate and nuanced data structures makes it indispensable in advancing machine learning applications, despite the trade-off in human interpretability.

1) Feedforward Neural Networks: Feedforward Neural Networks (FNNs) are a type of artificial neural network, where information flows only in one direction, namely from the input layer through the hidden layers to the output layer. FNNs are also known as multilayer perceptrons (MLPs) and are characterized by the absence of cycles or loops in the network structure. In an FNN, each neuron in a layer is connected to every neuron in the subsequent layer, forming a fully connected layer. This means that the output of each neuron in one layer serves as input to all neurons in the next layer. The connections between neurons are weighted, and each neuron applies an activation function to the weighted sum of its inputs to produce an output [5].

FNNs are powerful models that can approximate any continuous function to arbitrary precision, thanks to the universal approximation theorem. They have been successfully applied in various domains, including image classification, natural language processing, and financial forecasting. Training an FNN involves adjusting the weights and biases of the network using optimization algorithms such as gradient descent to minimize the difference between the predicted outputs and the true outputs [5].

2) Recurrent Neural Networks: Recurrent Neural Networks (RNNs) are a type of artificial neural network commonly used in tasks involving sequential data, such as speech recognition, natural language processing, and time series analysis. Unlike traditional neural networks where inputs and outputs are independent of each other, RNNs have connections that allow information to flow in a loop, enabling them to capture and utilize sequential dependencies in the data. The key feature of RNNs is their ability to maintain an internal memory or hidden state that allows them to process inputs in a sequential manner. At each time step, the output from the previous step is fed as input to the current step, allowing the network to retain information about the past and make predictions based on the context of the sequence. RNNs are composed of interconnected layers of artificial neurons, or nodes, with weighted connections between them. These connections allow the network to learn and update its parameters, such as weights and biases, through a process called backpropagation. During training, the network compares its output to the desired output and adjusts its parameters to minimize the error [11], [15]. The specific implementation of the recurrence block distinguishes several types of RNNs. RNNs are a helpful model when classifying Compton camera events because of the ability of the model to encode information about previous events into the evaluation of another event.

Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) are recurrent neural network (RNN) layers that offer advantages in capturing sequential information and handling long-term dependencies. They are widely used in various deep learning models, particularly in natural language processing, time series analysis, and other sequential data tasks. These layers possess gating mechanisms that control the flow of information through the network, enabling them to focus on relevant information and discard irrelevant or redundant information. This feature enhances the model's ability to retain essential information over time. In previous years, the gating mechanisms in GRU and LSTM layers have been used to help prevent overfitting by regulating the flow of information and controlling the network's capacity to memorize training data. This property is particularly beneficial when dealing with large and complex datasets, as it improves the model's generalization capability [6].

B. Translation to PyTorch

PyTorch and TensorFlow are both powerful, open-source deep learning frameworks, however, there are some general differences that might make one more appealing than the other for certain use cases. Previous works on this project used TensorFlow and Keras as ML library [1], [7], [8], [16]. In recent years, the percentage of research papers using PyTorch over TensorFlow has continued to grow reaching 75% of all new data science research papers using the PyTorch library [12]. PyTorch's dynamic computation graph and native Pythonic design can make it more intuitive for Python developers. This can lead to simpler and more readable code, which can be especially beneficial in complex multi-GPU settings where clarity is crucial.

1) PyTorch Distributed Data Parallel (DDP): Threading is a standard solution to carrying out parallel tasks allowing tasks to be distributed across multiple threads. A benefit of using PyTorch is implementing a similar technique to train different machine learning models in parallel across multiple GPUs [10]. DDP splits the input data batch to the number of available GPUs, passes it with a copy of model to each GPU, and runs parallel forward and backward passes on each GPU. Gradients from all GPUs are then synchronized and averaged, ensuring consistent weight updates across the entire model [10]. DDP in PyTorch uses multiprocessing to avoid well-known issues of threading in Python that arise due to the Global Interpreter Lock (GIL). The Global Interpreter Lock only allows one program thread to use the Python Interpreter, locking all of the others. This is the issue with threading since the GIL prevents perfect parallelism [4]. Implementing DDP will allow for faster training and better scaling as the training data size grows. DDP reduces the memory footprint by sharing parameters and ensures consistency across the model during training. Another benefit of implementing DDP into our research is that it provides a fault tolerance mechanism to handle failures during distributed training.

2) Training Model Across Single/Multiple GPU(s): DistributedDataParallel is a PyTorch module used for distributed training across multiple devices or machines [10]. DDP is designed to work with GPU and CPU modules where model is trained across multiple devices. The DDP wrapper requires that the underlying module supports parallel execution. The device type we train our model on is CUDA, which means our model was on a GPU device. The DDP wrapper takes a device_ids array argument which specifies how many devices to use for parallel training. If the array contains only one element, the training is done on a single GPU but if the array contains more than one element, the training is performed on multiple GPUs. To ensure that all tensors used by the model are on CUDA devices, we checked that all the model parameters are on CUDA and verified that the model inputs during training are also on CUDA devices [2].

C. Hardware and Software

We used the Graphics Processing Unit (GPU) clusters in the ada system in the UMBC High Performance Computing Facility (hpcf.umbc.edu) for our hyperparameter studies. The ada system has 3 distinct node types: four nodes with 8 NVIDIA RTX 2080 Ti GPUs each with 11 GB GPU memory; seven nodes with 8 NVidia Quadro RTX 6000 GPUs each with 24 GB of GPU memory; two nodes with 8 NVidia Quadro RTX 8000 GPUs each with 48 GB GPU memory. Each node has 384 GB of CPU memory (12 × 32 GB DDR4 at 2933 MT/s), except the two RTX 8000 nodes, which have 768 GB of CPU memory (12 × 64GB DDR4 at 2933 MT/s).

Networks built on ada were built using PyTorch v1.12.1 (https://pytorch.org/). We also used Pytoch Lightning v2.0.6 (https://lightning.ai/) which is an open-source Python library that provides a high-level interface for PyTorch. Moreover pandas v1.1.0 (https://pandas.pydata.org/) and numpy v1.25.1 (www.numpy.org) were also used to help preprocess the data. Finally, we used the matplotlib v3.5.1 (www.matplotlib.org) library to graph our results. Networks built on ada were built inside the python virtual environment package Anaconda3 (https://www.anaconda.com/) v4.8.3.

IV. Results

We trained the neural networks using a dataset derived from a Monte Carlo simulation, comprising 1,443,993 labeled data points. Each data point consists of 15 features which have been classified into 13 classes. These features encapsulate spatial coordinates, the distance between points, and energy levels for every interaction. An interaction is a grouping of three spatial coordinates and an energy level. Each row is either a triple, double-to-triple, or a false triple and consists of three interactions each. Our training data set only consisted of True Triples, Double-to-Triple scatter, and False events. For our training, we utilized datasets exposed to 150MeV beams at three varying intensity levels: 20kMU, 100kMU, and 180kMU, with higher kMU values indicating stronger dosage rates. We reformatted both our training and testing datasets for sequential processing. As a result, each 15-feature entry was transformed into a sequence of three interactions, each having five features: three spatial points, distance, and energy deposition. This data was then input into the neural network in sequences of three interactions. We allocated 20% of the data for validation purposes during the training process. [2].

We encountered a significant change when using the cross-entropy loss function in PyTorch. This function does not work with a data format called one-hot encoded labels. So, we did not use these labels during training. We made this specific change to make sure our training with PyTorch's cross-entropy loss function was efficient and accurate [2].

A. Hyper-parameter Study

Previous research explored various networks, such as fully connected neural networks and RNNs, for the classification task [1], [7], [8], [16]. In this work we utilized similar model structures in PyTorch and tried to improve the performance of our model with hyper-parameter tuning. We begin by examining the number of epochs, the batch size, and the learning rate. We then explore the number of layers, neurons,

Hyperparameter	*Value*
Layers	256
Neurons	256
Batch Size	8192
Learning Rate	1e-3
Train/Validation	0.8/0.2
Dropout	0.45
Inter-activation	leakyrelu
Clip Gradient	0

TABLE I: Variable Model Parameters.

and the dropout rate to determine a promising configuration for the network [2]. The varying hyperparameters that are tuned throughout this study can be seen in Table I.

1) Number of Layers: When starting this study, we observe the parameters used for the dense models runs of previous TensorFlow models [16]. We first investigated the number of layers, starting with 64, 128, and 256 layers. These runs stagnated at roughly 7% accuracy with little to no improvement with increasing epochs. We then tested a model with only 16 layers, which saw a significant increase in accuracy from 7% to 49.7% , indicating that fewer layers leads to higher accuracy in our fully connected model. We then conducted 2048 epoch runs with 1–16 layers. We found that if there were too few layers, the model would become severely over fitted. 11 layers produced the best accuracy while limiting overfitting [2].

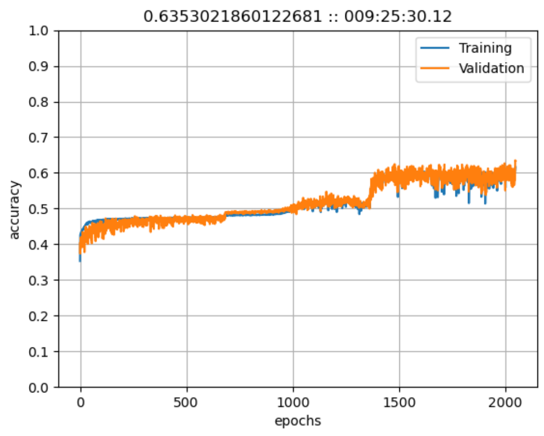
2) Learning rate: After finding that a lower number of dense layers produced better results, the next step was to adjust our learning rate hyperparameters. We trained 2, 4, 6, and 11 layer dense models with constant hyperparameters defined in Table II. We utilized learning rate scheduler to adjust the learning rate throughout the training process, which can lead to faster convergence and better performance. PyTorch offers various scheduling strategies, such as StepLR, which modifies the learning rate at specific epoch intervals, and ExponentialLR, which decays the learning rate exponentially over epochs. Others, like ReduceLROnPlateau, adjust the rate based on the recent model performance, decreasing it when a plateau is detected. In this work we used StepLR strategy, with an initial learning rate of 1e-3, which was multiplied by of 1e-1, 3e-1, and 5e-1 every 682 epochs and 341 epochs. From this experiment, we found better results from changing the learning rate every 341 epochs [2]. When we multiplied the learning rate by 1e-1, 3e-1, and 5e-1, we achieved an accuracy of 63.9%, 63.5%, and 59.8% and 5e-1 had an accuracy of 59.8%. Decreasing the learning step from 682 to 341 improve all models independent of learning change by a couple of percentage points, as shown in Figure 2.

3) Batch Size: Larger batch sizes can improve model performance by providing more stable weight updates, efficient parallel processing, and better generalization. However, extremely large batch sizes can lead to memory issues and slower convergence.

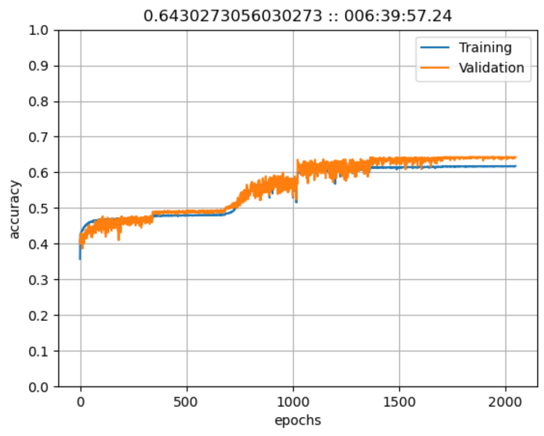
To determine the best batch size for the data we trained models using batch sizes of 8192, 16384, 32768, and 65536 as shown in Table III. Based on our training of these model

Hyperparameter	*Value*
Learning Rate	1.0e-3
Train/Validation	0.8/0.2
Batch Size	8192
Neurons	256
Number of Layers	11
Dropout	0.45
Inter-activation	leakyrelu
Clip Gradient	1.0e-3

TABLE II: Hyperparameters in learning rate model.



(a) 11 layer Dense model with 3e-1 learning change and 682 learning step



(b) 11 layer Dense model with 3e-1 learning change and 341 learning step

Fig. 2: Graphs of model accuracy.

we concluded that a batch size of 8192 has the best accuracy of any of the models with the least training time. As the batch size increase the training time increased while the accuracy of the model significantly decrease [2].

B. RNN with LSTM and GRU layers

In RNN training with LSTM layers, we tested our model with 2, 4, 16, and 128 LSTM layers. Both 16 and 128 LSTM layers performed poorly with accuracy not more than 7.73%.

Batch Size	*Val. Accuracy*
8192	63.96%
16384	63.89%
32768	52.74%
65536	50.32%

TABLE III: Batch size on 11-layer dense model accuracy.

For 4 LSTM layers, we obtained accuracies of 76.0%, 66.5%, 63.7%, and 56.4% using L2 regularization of 1e-2 and 1e-3, while for 2 LSTM layers, we obtained accuracies of 56.6%, 54.1%, 63.7%, and 63.3% using L2 regularization of 1e-2 and 1e-4. During our training, we also observed the perfomance of our model using 2, 4, and 6 GRU layers. For 6 GRU layers, the runs produced a 7.7% accuracy with 16 dense layers and a 7.6% accuracy with 64 dense layers. For 4 GRU layers, we achieved a 55.1% accuracy with 16 dense layers and 8.4% accuracy with 64 dense layers. The best performed RNN training was the 2 GRU layers, which produced a 64.4% accuracy with 10 dense layers and a 66.0% accuracy with 16 dense layers.

C. Multiple GPU training

In our experiments, we undertook the training of two RNN models across multiple GPUs to explore the potential benefits of distributed training. Table IV (a) shows the result of training the RNN model with 4 LSTM layers, followed by two fully connected layers with ReLU activation function, for 1024 epochs on single and multiple GPUs. Similarly, Table IV (b) demonstrates the results of single and multi-GPU training of the other RNN model, which utilizes 4 GRU layers coupled with the same fully connected layers in the LSTM model.

As is shown Table IV, while using the power of multiple GPUs significantly reduced our training times, leading to more efficient model iterations and quicker experimentation cycles, there was a slight decrease in both training and validation accuracies. This minor reduction in accuracy could be attributed to factors like the effective batch size, different gradient aggregation dynamics, and the complexities introduced by parallel processing. The Distributed Data Parallel (DDP) strategy in PyTorch evenly divides the total batch across multiple GPUs, resulting in what is termed as ”mini-batches”. For example, with a total batch size of 8019 and the use of 4 GPUs, each GPU processes 2046 inputs, making the mini-batch size 2048. Moreover, we proportionally increased both the total batch size and the learning rate (LR) with each doubling of GPU count. This ensures that each GPU in a multi-GPU setup effectively maintains the same batch size and learning rate as in single GPU training. However, these adjustments did not yield the anticipated improvements in model performance.

To have a better demonstration of multi-GPU training, we provide a visual representation of how the validation accuracy of our RNN model varies with the use of different numbers of GPUs. We plot the accuracy graph for the two LSTM and GRU models, with 8192 batch size and 1.0e-3 learning rate, for 1024 epochs on different numbers of GPUs. As shown

# GPUs	*Total batch Size*	*Learning rate*	*Training accuracy*	*Validation accuracy*	*Time (minutes)*
1	8192	1.0e-3	73.40%	63.14%	453.84
2	8192	1.0e-3	67.51%	63.45%	205.93
2	16384	2.0e-3	68.14%	60.56%	191.44
4	8192	1.0e-3	63.39%	61.47%	163.53
4	16384	2.0e-3	63.32%	60.94%	171.18

(a) RNN with 4 LSTM layers multi-GPU training time for 1024 epochs

# GPUs	*Total batch Size*	*Learning rate*	*Training accuracy*	*Validation accuracy*	*Time (minutes)*
1	8192	1.0e-3	74.21%	66.45%	440.01
2	8192	1.0e-3	70.13%	65.88%	204.49
4	8192	1.0e-3	65.17%	62.89%	159.39
4	16384	2.0e-3	63.59%	59.92%	166.99

(b) RNN with 4 GRU layers multi-GPU training time for 1024 epochs

TABLE IV: Comparison of RNN with LSTM and GRU layers for multi-GPU training.

in Figure 3, the best overall validation accuracy was obtained when using 2 GPUs in both models.

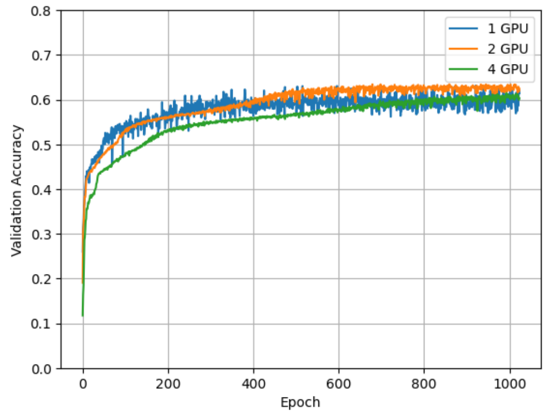
Nonetheless, the accelerated training process presents a compelling advantage. It demonstrates that, with the right optimization strategies and meticulous hyperparameter tuning, multi-GPU training can be an invaluable tool for deep learning practitioners, especially when time-to-solution is paramount. As we further refine our distributed training techniques, we anticipate narrowing the accuracy gap, all while reaping the benefits of reduced training duration.

V. Conclusions

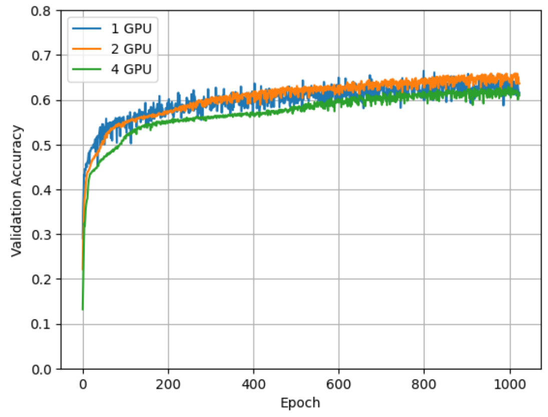
Our research focuses on the methodology of defining and training DNNs in PyTorch, which provides the advantage of dynamic computational graphs. In this study, we explored various networks, including Dense, LSTM, and GRU layers, for classification to optimize the performance of recurrent neural networks (RNNs).

To determine the optimal network configurations, we conducted an extensive hyper-parameter study. Among those hyper-parameters, the number of epochs, batch size, learning rate, learning-rate scheduler, and number of layers had a noticeable impact on the models' performance. Initial tests with higher layer counts (64, 128, and 256) resulted in a stagnant accuracy of around 7%. However, a model with 16 layers exhibited a notable increase in accuracy, and 11 layers were found to be optimal, achieving the best accuracy while effectively preventing overfitting. Furthermore, the learning rate's adjustment, especially when using the StepLR strategy, proved beneficial in enhancing model accuracy. Additionally, we discovered that a batch size of 8192 yielded the highest accuracy while minimizing training time, making it the favored option for our model. The influence of clip gradient and dropout rate on peak accuracy was marginal.

In the realm of recurrent neural networks (RNNs), both LSTM and GRU layers were explored. While models with 16 and 128 LSTM layers underperformed, achieving a mere



(a) 4 LSTM layers + 2 Dense layers



(b) 4 GRU layers + 2 Dense layers

Fig. 3: Comparison of validation accuracy for the RNN model trained across multiple GPUs.

7.73% accuracy, the 4-layer LSTM configuration yielded accuracies up to 76.0% with specific L2 regularization settings. On the other hand, GRU-based models showcased the best performance with 2 layers, achieving a 66.0% accuracy with 16 dense layers. This study underscores the significance of layer configurations and regularization in determining RNN performance.

Moreover, in our exploration of multi-GPU training for RNN models, we found that while leveraging multiple GPUs considerably expedited the training process, it introduced a slight decline in both training and validation accuracies. More specifically, training on two GPUs was twice faster than a single GPU with almost the same validation accuracy, however, run-time on four GPUs is roughly 20% faster than two GPUs with around 3% lower validation accuracy. Despite making adjustments to the total batch size and learning rate in alignment with the DDP strategy, the expected performance enhancement remained elusive so far. However, the rapidity advantage of multi-GPU training cannot be overlooked.

Acknowledgments

This work is supported by the NSF-grant "REU Site: Online Interdisciplinary Big Data Analytics in Science and Engineering" from the National Science Foundation (grant no. OAC–2050943). Co-author Ren acknowledges support from the NIH-grant R01–CA279013. Co-author Cham additionally acknowledges support as HPCF RA. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258, CNS–1228778, OAC–1726023, and CNS–1920079) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See hpcf.umbc.edu for more information on HPCF and the projects using its resources.

References

[1] Alina M. Ali, David Lashbrooke, Rodrigo Yepez-Lopez, Sokhna A. York, Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Towards optimal configurations for deep fully connected neural networks to improve image reconstruction in proton radiotherapy. Technical Report HPCF–2021–12, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2021.

[2] Kaelen Baird, Sam Kadel, Brandt Kaufmann, Ruth Obe, Yasmin Soltani, Mostafa Cham, Matthias K. Gobbert, Carlos A. Barajas, Zhuoran Jiang, Vijay R. Sharma, Lei Ren, Stephen W. Peterson, and Jerimy C. Polf. Enhancing real-time imaging for radiotherapy: Leveraging hyperparameter tuning with PyTorch. Technical Report HPCF–2023–12, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2023.

[3] Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Deep residual fully connected neural network classification of Compton camera based prompt gamma imaging for proton radiotherapy. *Front. Phys.*, 11:903929, 2023.

[4] Jason Brownlee. ThreadPool and the global interpreter lock (gil). In *SuperFast Python*. 2021. https://superfastpython.com/threadpool-gil/.

[5] Rene Y. Choi, Aaron S. Coyner, Jayashree Kalpathy-Cramer, Michael F. Chiang, and J. Peter Campbell. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science & Technology*, 9(2):14–14, 02 2020.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Joseph Clark, Anaise Gaillard, Justin Koe, Nithya Navarathna, Daniel J. Kelly, Matthias K. Gobbert, Carlos A. Barajas, and Jerimy C. Polf. Sequence-based models for the classification of Compton camera prompt gamma imaging data for proton radiotherapy on the GPU clusters taki and ada. Technical Report HPCF–2022–12, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2022.

[8] Joseph Clark, Anaise Gaillard, Justin Koe, Nithya Navarathna, Daniel J. Kelly, Matthias K. Gobbert, Carlos A. Barajas, and Jerimy C. Polf. Multi-layer recurrent neural networks for the classification of Compton camera based imaging data for proton beam cancer treatment. In *9th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT 2022)*, in press (2022).

[9] Zhuoran Jiang, Jerimy C. Polf, Carlos A. Barajas, Matthias K. Gobbert, and Lei Ren. A feasibility study of enhanced prompt gamma imaging for range verification in proton therapy using deep learning. *Phys. Med. Biol.*, 68(7):075001, 2023.

[10] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. PyTorch distributed: Experiences on accelerating data parallel training, 2020.

[11] Javaid Nabi. Recurrent Neural Networks (RNNs). In *Towards Data Science*. 2019. https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85.

[12] Ryan O'Connor. PyTorch vs TensorFlow in 2023. In *Assembly AI*. 2023. https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/.

[13] Jerimy C. Polf, Carlos A. Barajas, Stephen W. Peterson, Dennis S. Mackin, Sam Beddar, Lei Ren, and Matthias K. Gobbert. Applications of machine learning to improve the clinical viability of Compton camera based in vivo range verification in proton radiotherapy. *Front. Phys.*, 10:838273, 2022.

[14] Jerimy C. Polf and Katia Parodi. Imaging particle beams for cancer treatment. *Phys. Today*, 68(10):28–33, 2015.

[15] Chi-Feng Wang. The vanishing gradient problem: The problem, its causes, its significance, and its solutions. In *Towards Data Science*. 2019. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[16] Sokhna A. York, Alina M. Ali, David C. Lashbrooke Jr, Rodrigo Yepez-Lopez, Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Promising hyperparameter configurations for deep fully connected neural networks to improve image reconstruction in proton radiotherapy. In *2021 IEEE International Conference on Big Data (Big Data 2021)*, pages 5648–5657, 2021.