UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Practical Work 3:
High-Performance Computing Aspects of Deep Learning

Deep Learning

João Valério

joao.agostinho@estudiantat.upc.edu

31/05/2023

Index

1. INTRODUCTION

The field of deep learning has witnessed significant advancements in recent years due to the availability of large-scale datasets and the rapid growth in computing power. High-Performance Computing (HPC) plays a vital role in enabling the efficient execution of deep learning algorithms, resulting in improved model accuracy and overall performance. Understanding the intricacies of HPC in deep learning is crucial for researchers, practitioners, and organizations to harness its potential in accelerating training, optimizing model performance, and achieving remarkable results across various domains.

This report presents a series of practical exercises aimed at providing hands-on experience and insights into optimizing deep learning models using HPC techniques. The exercises are structured as follows:

Exercise 1: This exercise explores the impact of different learning rates on model training using the GradientDescentOptimizer. By experimenting with various learning rates and observing their effects on model convergence and accuracy, the objective is to determine the optimal learning rate. Additionally, alternative descent methods will be investigated.

Exercise 2: In this exercise, the convergence rates of a deep neural network will be analyzed using different optimizers and learning rates. The widely-used MNIST dataset will be used for this analysis. The deep neural network architecture consists of a first convolutional layer with 32 features per 5x5 patch, a second convolutional layer with 64 features per 5x5 patch, and a densely connected layer with 1024 neurons processing 64 7x7 images. The model includes a dropout rate of 50%. By plotting and comparing the convergence rates achieved with different optimizers and learning rates, valuable insights into their impact on model training can be gained.

Exercise 3: The objective of this exercise is to maximize the accuracy rate of the deep learning model. A hint suggests that rearranging the way batches are defined can lead to accuracies above 99%. By implementing this hint and exploring various batch configurations, the aim is to achieve the highest possible accuracy with the given model architecture and dataset.

Exercise 4: This exercise focuses on reducing the training time of the deep learning model introduced in Exercise 3. While the previous examples utilized a single GPU, the goal is to leverage the power of multi-GPU systems to expedite the training process. Specifically, the possibility of utilizing all four GPU devices on a node of the P9 cluster or any other multi-GPU system will be explored. By measuring and comparing training times using 1, 2, and 4 GPUs, the effectiveness of parallelization in reducing training time can be evaluated. Furthermore, the implementation details involved in utilizing multiple GPUs for deep learning training will be discussed.

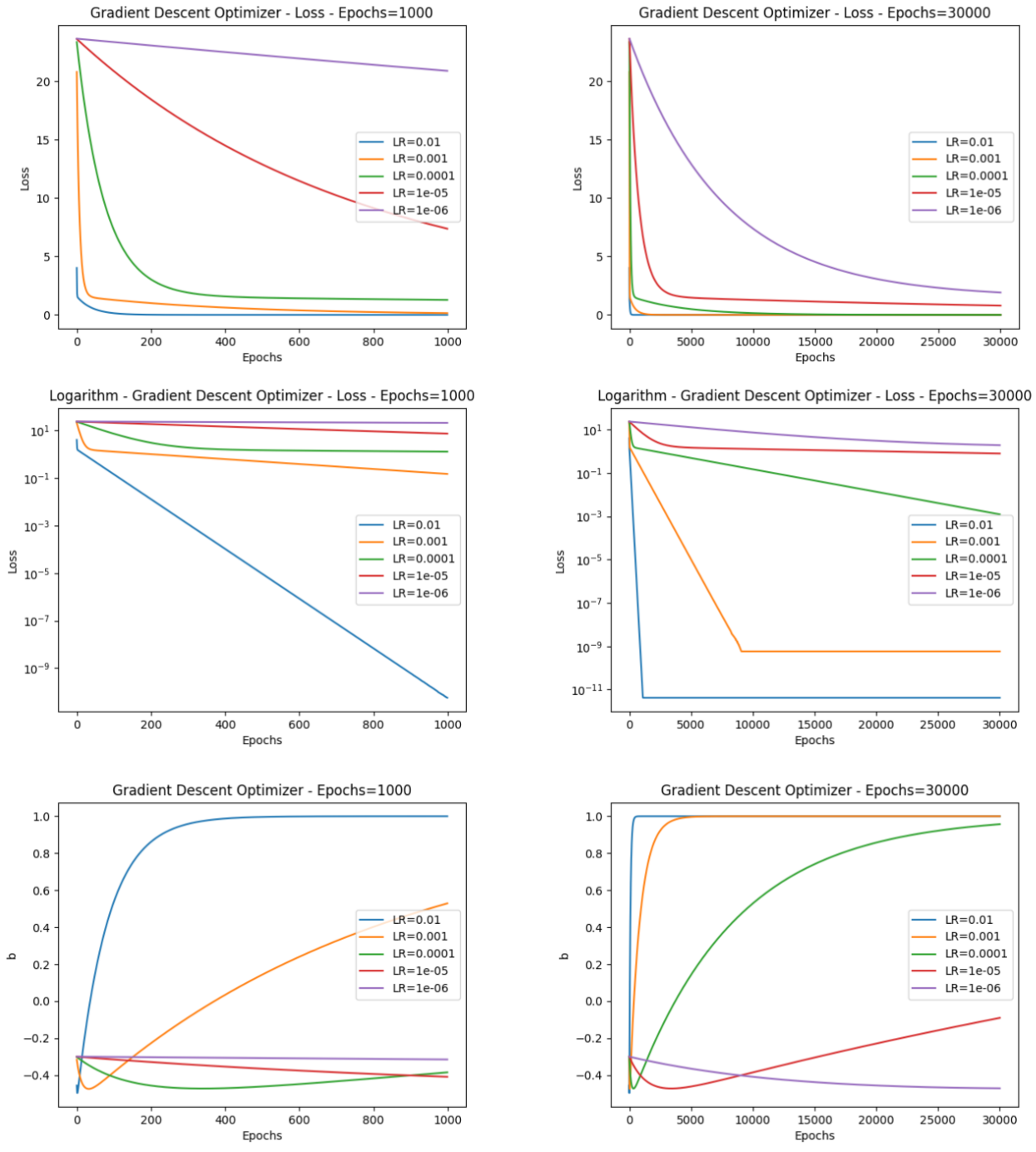
These exercises aim to provide practical experience in optimizing deep learning models, exploring different learning rates and optimizers, increasing model accuracy, and harnessing the power of multi-GPU systems. The insights and results obtained from these exercises will enhance understanding of the HPC aspects in deep learning and their practical implications in real-world scenarios.

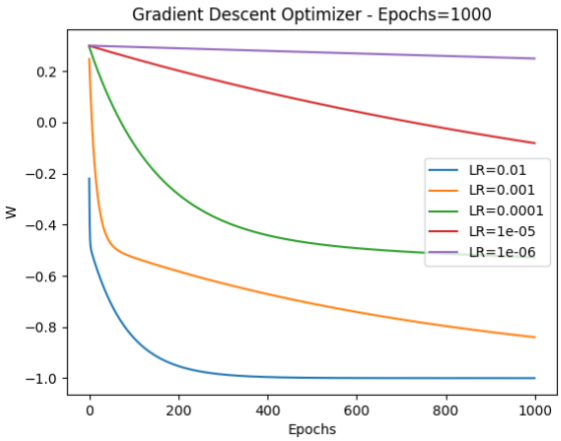
2. EXERCISE 1

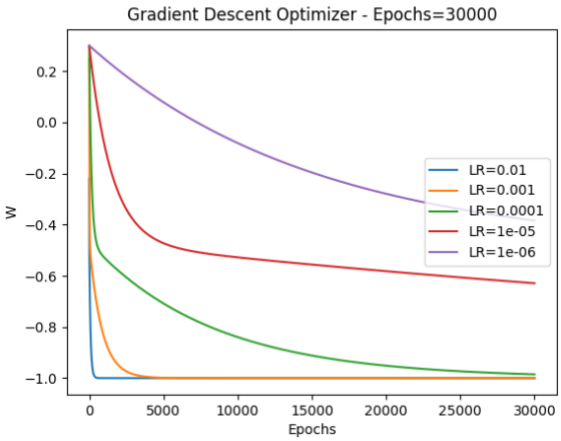
a. Gradient Descent Optimizer With Different Learning Rates

Firstly, the influence of the learning rate value on the Gradient Descent Optimizer (GDS) is examined. The objective is to analyze the loss and the linear function's parameters (W and b) throughout the learning process. Consequently, a range of learning rate values is tested, specifically 1e-2, 1e-3, 1e-4, 1e-5, and 1e-6, on different numbers of epochs: 1000 and 30000.

Figure 2.a.1. - Loss, W and b evolutions.







Based on the obtained results, it is evident that the learning rate significantly influences the evolutions observed in the learning curves. Consequently, a lower learning rate leads to a slower convergence process, resulting in less pronounced movements of the weights within the weight space.

Furthermore, a visual examination of the convergence patterns reveals that 1000 epochs provide sufficient observation for the convergences represented by the blue line (0.01 learning rate) and the yellow line (0.001 learning rate). However, 3000 epochs fail to capture the convergence represented by the purple line (1e-6 learning rate), due to the extremely slow learning process.

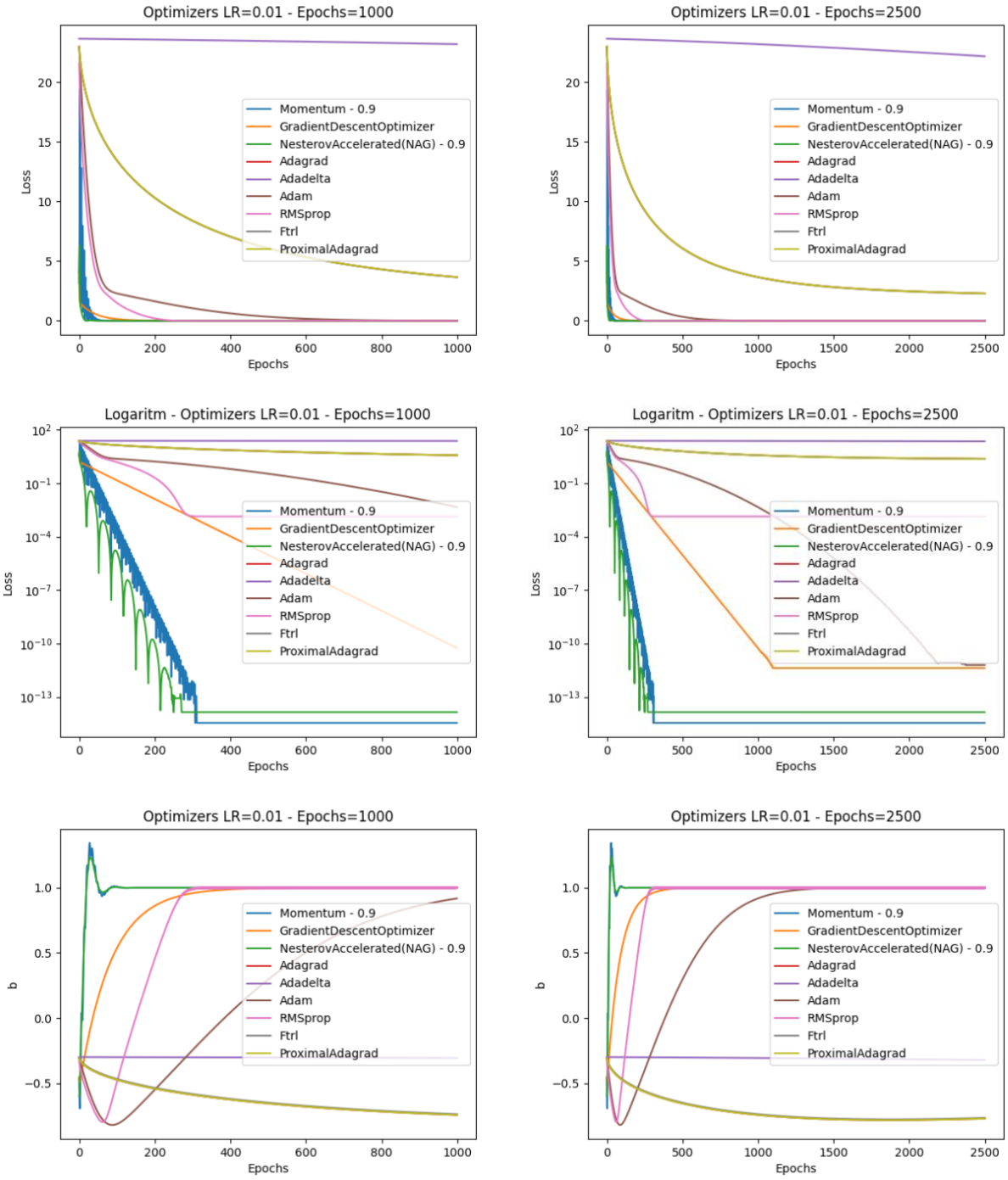
Nonetheless, since the learning rate of 0.01 exhibits the convergence point with the lowest loss associated, there are no discernible benefits in employing a learning rate lower than 0.01. This observation stems from the relative simplicity of the problem, where a learning rate of 0.01 is enough to facilitate the convergence towards the global minima.

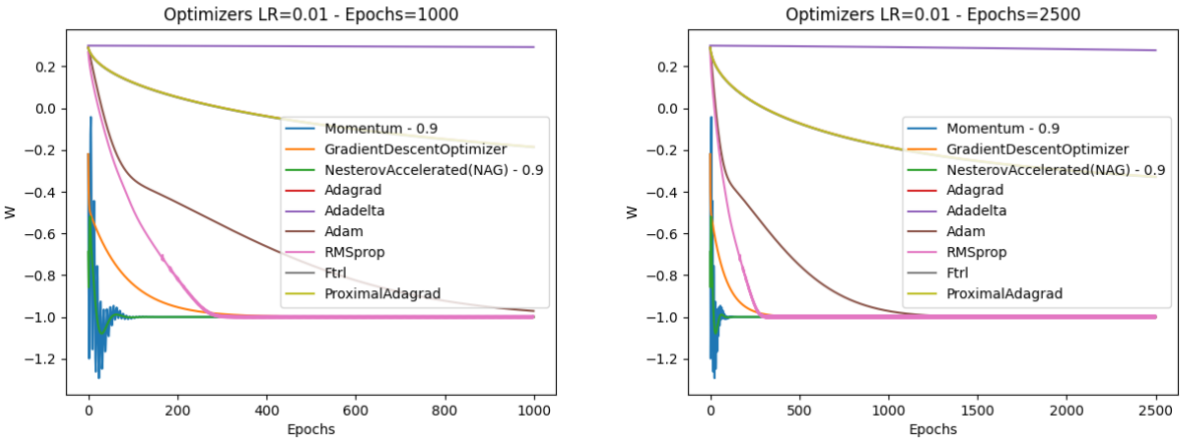
Regarding the linear function parameters, W and b, their evolutions exhibit similar underlying configurations influenced by the learning rate. However, it is noteworthy that while W evolves along a consistent direction within the spatial dimensions, b initially deviates towards a different direction, away from the localization of the global minima. This latter pattern becomes more pronounced when a lower learning rate is employed.

b. Optimizers with Learning Rate = 0.01

Following the previous study, it has been determined that fixing the learning rate at 0.01 yields the optimal value. Consequently, a series of optimizers are being examined to understand their distinct behaviors over 1000 and 2500 epochs. In order to sustain the analysis, the loss, logarithmic loss, and linear function parameters are scrutinized, as was done in chapter a. Notably, the selected optimizers for evaluation are Momentum, Gradient Descent, Nesterov Accelerated, Adagrad, Adadelta, Adam, RMSprop, Ftrl, and Proximal Adagrad. It is worth mentioning that Proximal Gradient Descent is excluded from the assessment since its results were found to be similar to those of Gradient Descent, thereby exhibiting no noticeable difference.

Figure 2.b.1. - Loss, W and b evolutions.





Based on the observed loss values, it is evident that all the models exhibit signs of convergence. However, not all of them surpass the results achieved by the gradient descent optimizer in the previous chapter.

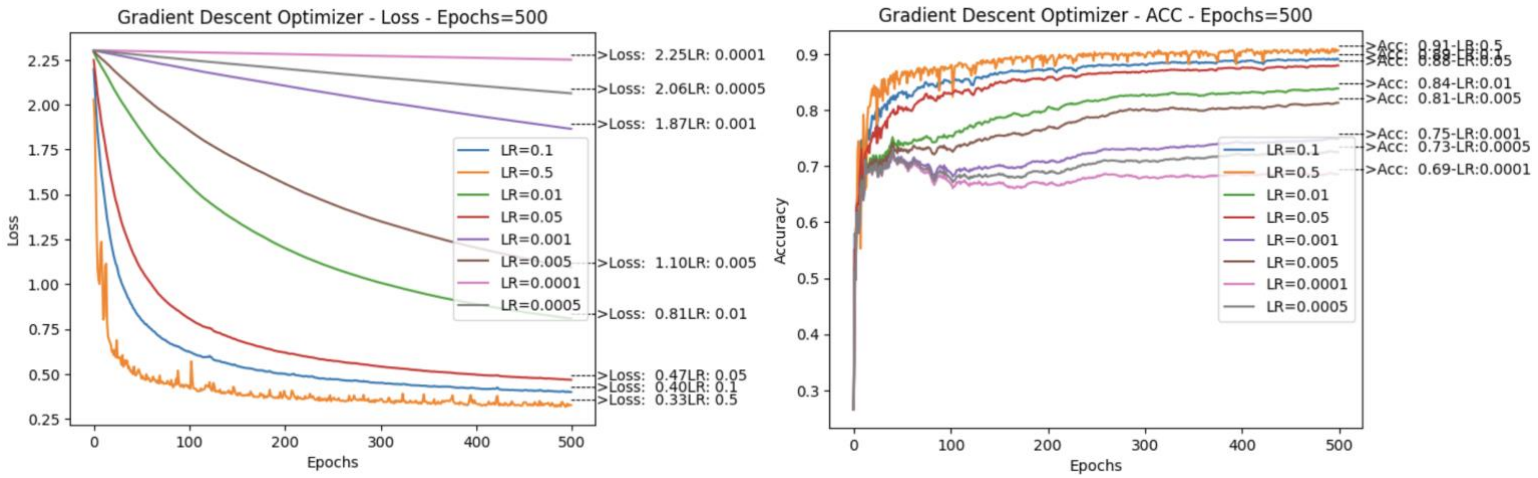
Upon analyzing the logarithmic scale, it can be inferred that the top-performing models, in descending order, are Momentum, Nesterov Accelerated, and Gradient Descent. It is noteworthy that while both Momentum and Nesterov Accelerated demonstrate superior convergence compared to Gradient Descent, their graphical representations exhibit higher instability and lack smoothness. Furthermore, among these two optimizers, Momentum displays fewer oscillations, which suggests a potentially more optimal convergence.

Finally, it is worth emphasizing that both Momentum and Nesterov Accelerated optimizers exhibit faster convergence to their respective minima. Consequently, the attainment of the optimal parameters for the function is also expedited.

3. EXERCISE 2

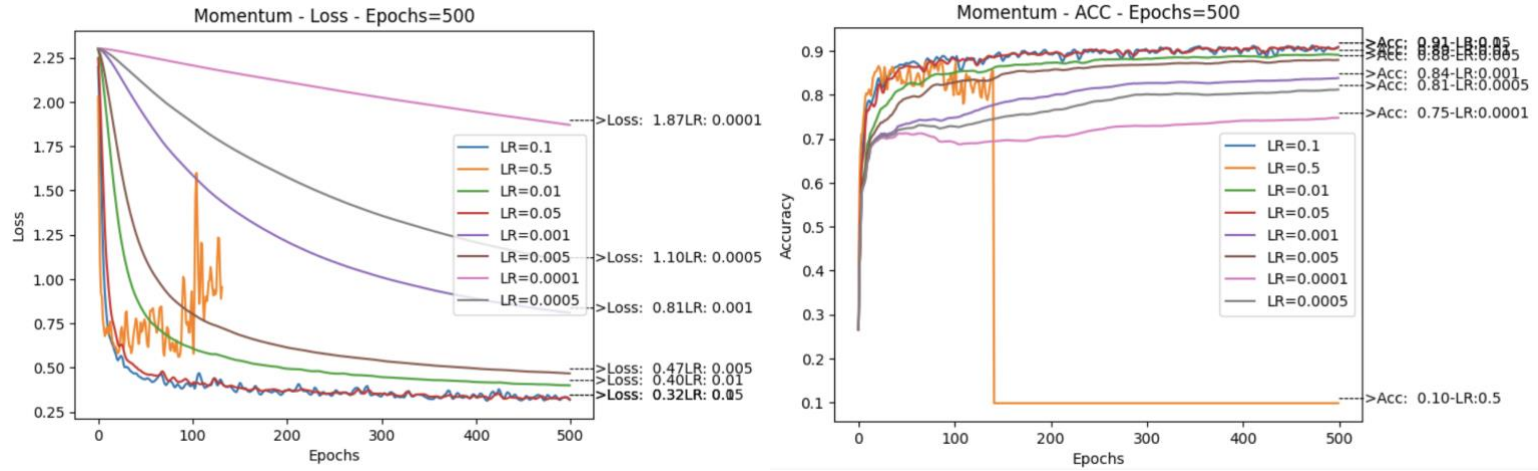
In light of the newly provided network, experiments were undertaken to examine the progress of loss and accuracy throughout the training procedure. Accordingly, the optimizers employed bear resemblance to those discussed in chapter 2.b. Nevertheless, in this instance, a diverse set of learning rates were tested, namely: 0.1, 0.5, 0.01, 0.05, 0.001, 0.005, 0.0001, and 0.0005, while maintaining a constant number of epochs at 500.

Figure 3.1. - Loss and accuracy evolutions.



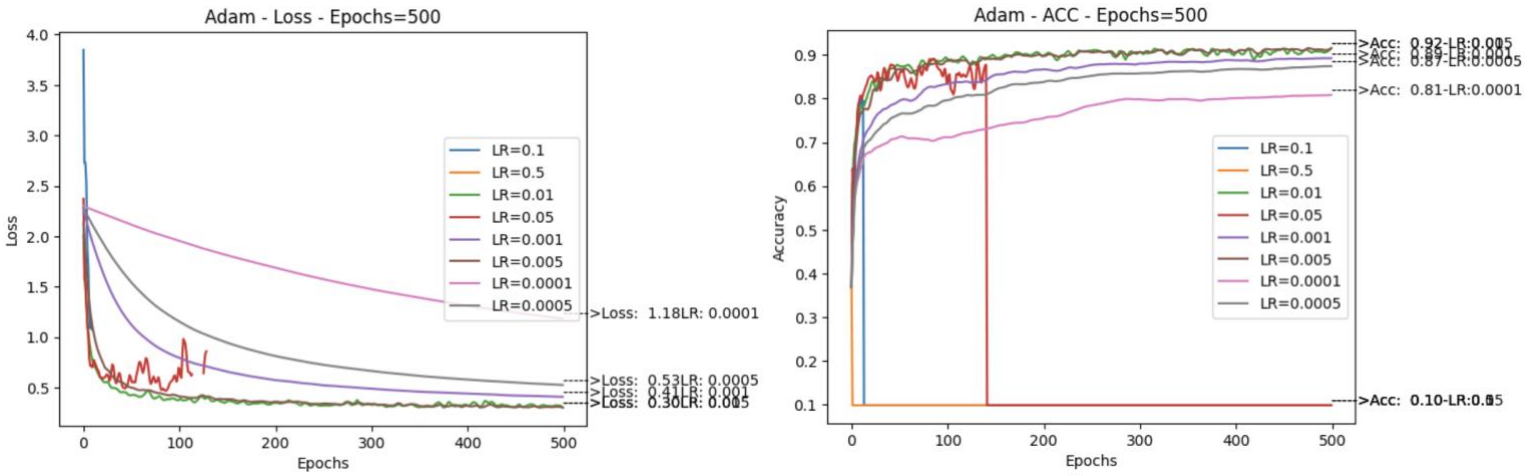
In the Gradient Descent Optimizer the best result was achieved with a 0.5 learning rate, with a loss of 0.33 and 0.91 accuracy.

Figure 3.2. - Loss and accuracy evolutions.



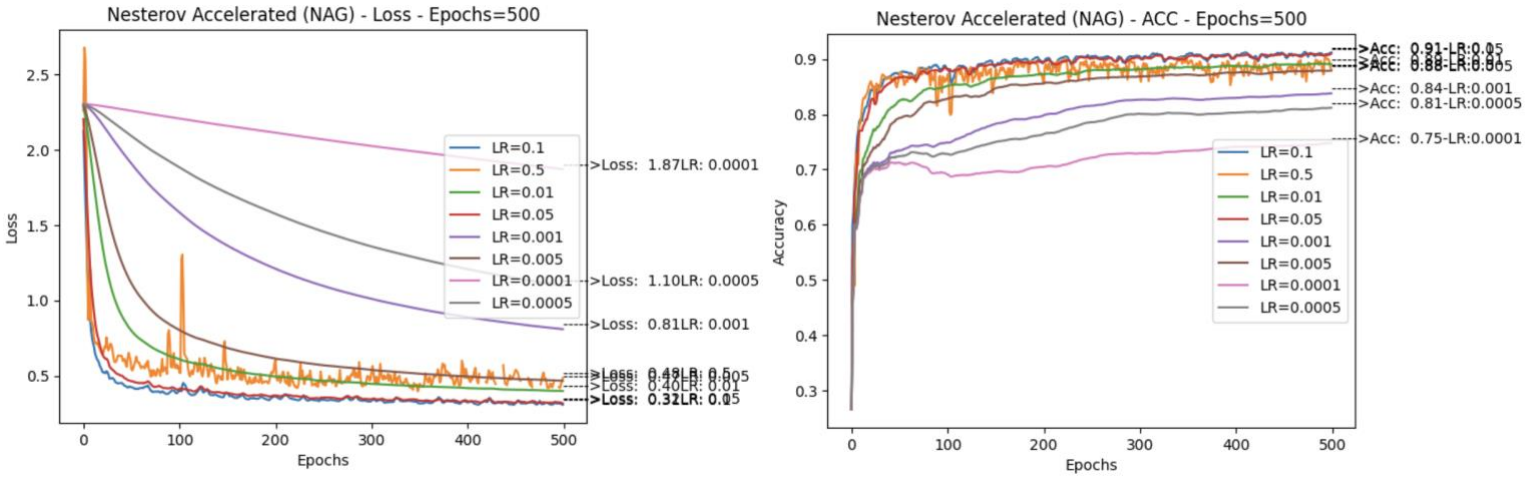
In the Momentum Optimizer the best result was achieved with a 0.05 and 0.1 learning rates, with a loss of 0.32 and 0.91 accuracy both.

Figure 3.3. - Loss and accuracy evolutions.



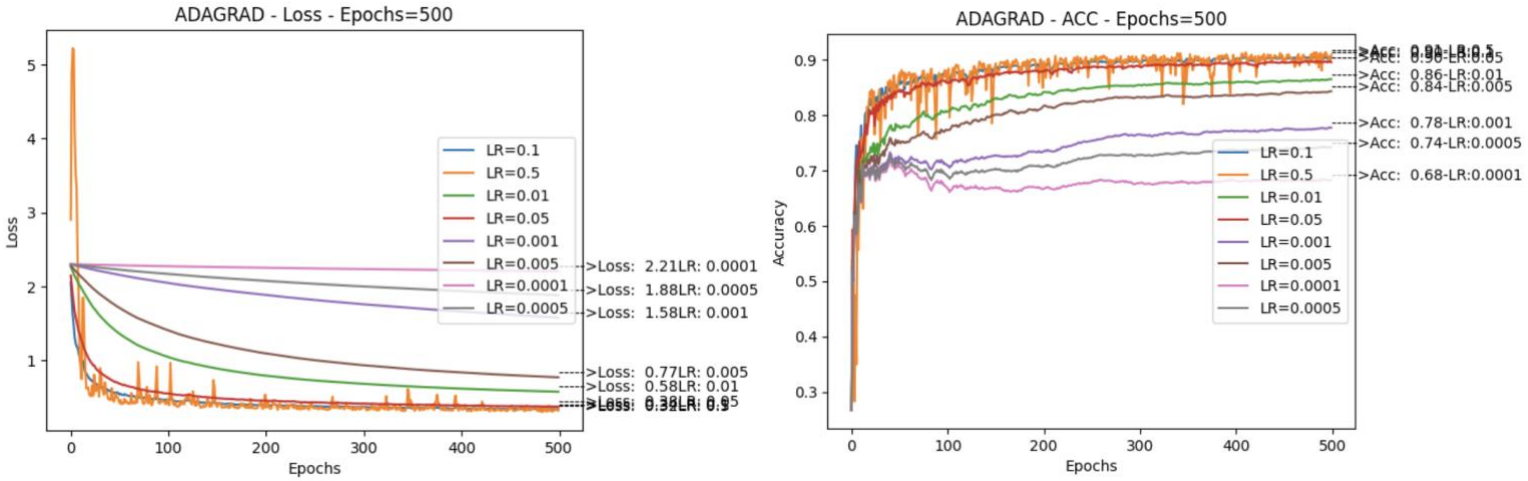
In the Adam Optimizer the best result was achieved with a 0.005 and 0.01 learning rates, with a loss of 0.30 and 0.92 accuracy both.

Figure 3.4. - Loss and accuracy evolutions.



In the Nesterov Accelerated Optimizer the best result was achieved with a 0.1 learning rate, with a loss of 0.31 and 0.91 accuracy.

Figure 3.5. - Loss and accuracy evolutions.



In the Adagrad Optimizer the best result was achieved with a 0.1 and 0.5 learning rates, with a loss of 0.32 and 0.91 accuracy both.

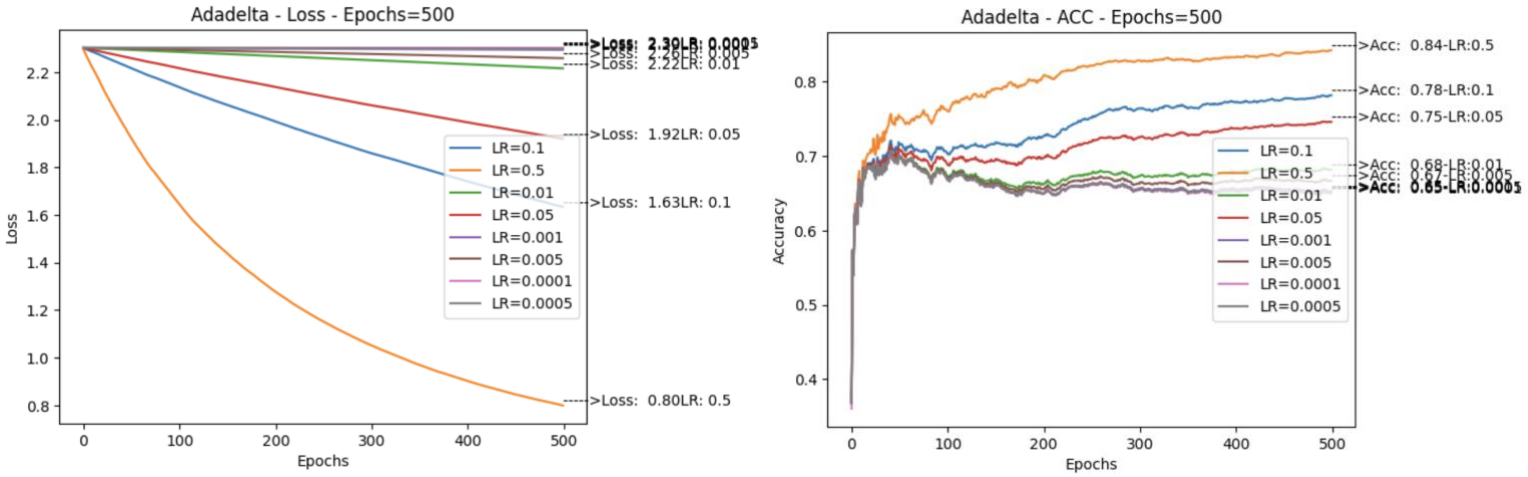
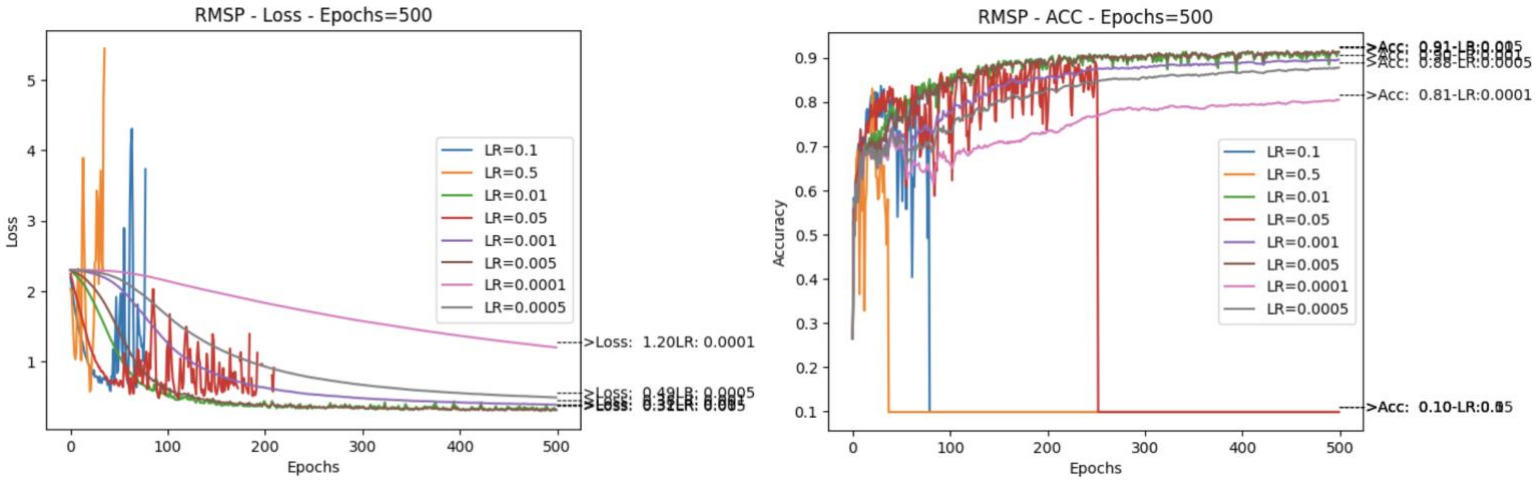


Figure 3.6. - Loss and accuracy evolutions.

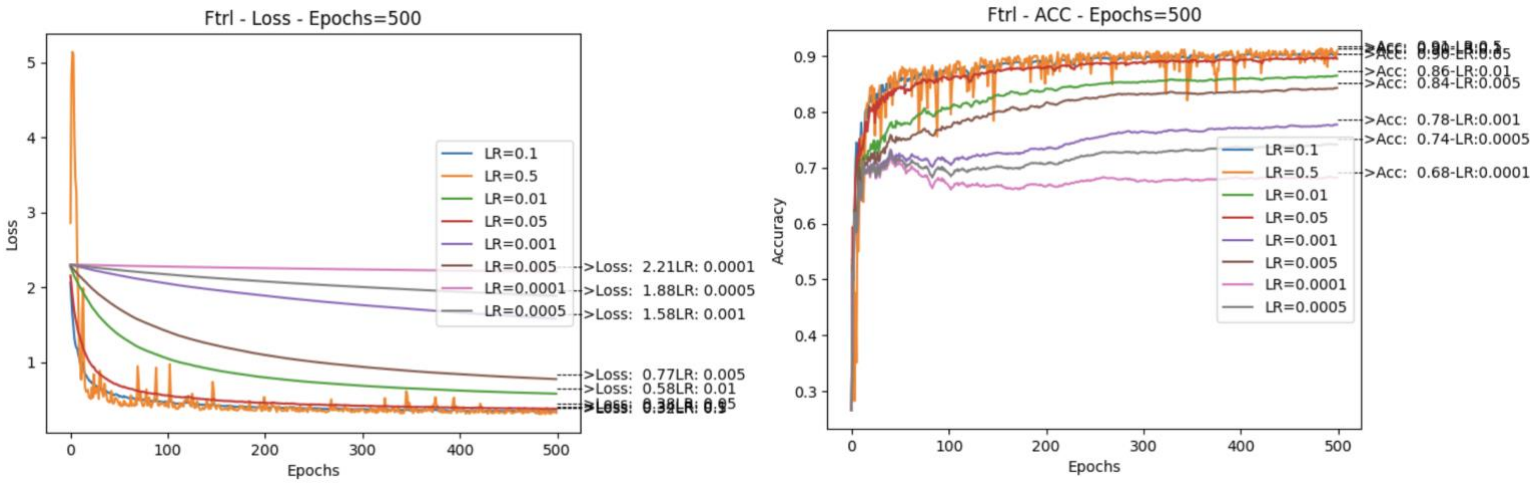
In the Adadelta Optimizer the best result was achieved with 0.5 learning rate, with a loss of 0.80 and 0.84 accuracy.

Figure 3.7. - Loss and accuracy evolutions.



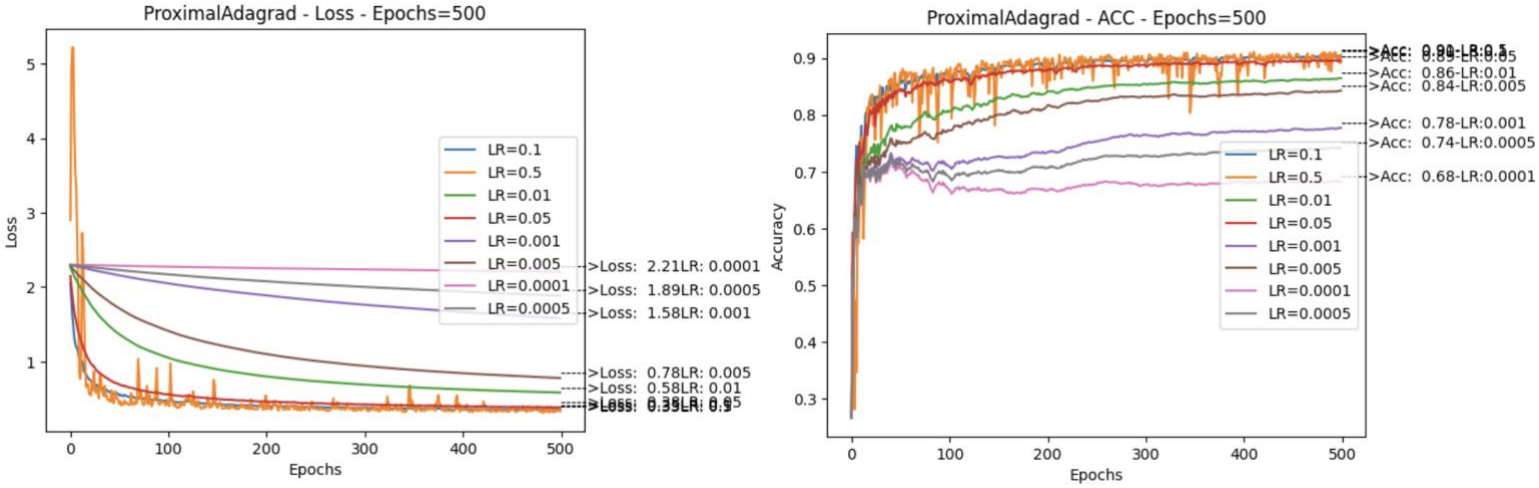
In the RMSP Optimizer the best result was achieved with 0.01 and 0.005 learning rates, with a loss of 0.32 and 0.91 accuracy both.

Figure 3.8. - Loss and accuracy evolutions.



In the Ftrl Optimizer the best result was achieved with 0.1 and 0.5 learning rates, with a loss of 0.32 and 0.91 accuracy both.

Figure 3.9. - Loss and accuracy evolutions.



In the Proximal Adagrad Optimizer the best result was achieved with 0.1 and 0.5 learning rates, with a loss of 0.33 and 0.91 accuracy both.

In general, excluding the Adadelta Optimizer, all the models demonstrated the capability to attain a loss value below 0.34 and an accuracy exceeding 0.90. Notably, the Adam Optimizer yielded the most favorable outcomes, with a loss of 0.30 and an accuracy of 0.92, particularly when utilizing learning rates of 0.005 and 0.01.

4. EXERCISE 3

In the current exercise, the objective is to enhance the accuracy rate of the provided new network to the greatest extent possible. Consequently, the approach to be employed involves redefining the batch composition methodology. Furthermore, given the demonstrated superiority of the Adam Optimizer in the previous chapter, it has been chosen as the optimizer for the current application.

Initially, by executing the provided basic code, an accuracy of 96.1% was achieved after 1000 epochs.

As mentioned earlier, the primary focus of the strategy lies in the configuration of the batches. Consequently, several different batch sizes were tested, spanning 5000 epochs, while maintaining either the default learning rate of 1e-4. Subsequently, the accuracy attained at the best epoch is presented in the following table:

Table 4.1. - Accuracy values.

Batch Size	150	100	50	25	10	8	6	4
Fixed Learning Rate	1e-4							
Accuracy [%]	94.1	95.0	95.5	96.2	**97.3**	**97.3**	96.9	96.4

Based on the obtained results, it is evident that the model's performance exhibits significant improvement by adjusting the batch size. Consequently, to leverage this observation, a randomized batch size arrangement is implemented for each epoch.

Table 4.2. - Accuracy values.

Batch Size	150	100	50	25	10	8	6	4
Fixed Learning Rate	1e-4							
Accuracy [%]	**98.9**	**98.9**	98.6	98.3	97.7	97.4	97.4	95.3

Hence, based on the obtained results, it can be concluded that the randomization of batches yields a noteworthy enhancement. However, this approach proves more effective when employed with larger batch sizes, such as 150 or 100, owing to the size and complexity of the associated data.

Consequently, an exponential decay mechanism for the learning rate is introduced. The initial value for the learning rate is set at 1e-4, which coincides with the default value used in the provided code. A decay rate of 0.95 is applied specifically for the optimal batch sizes of 150 and 100.

Table 4.3. - Accuracy values.

Batch Size	150	100
Exp. Decay Learning Rate	1e-4	
Accuracy [%]	99.0	99.0

While the improvement in the final accuracy may not be substantial, there is a discernible enhancement in the model's performance. Consequently, it has been determined to make adjustments to the learning rate in both configurations.

Table 4.4. - Accuracy values.

Batch Size	150								
Exp. Decay Learning Rate	2e-4	3e-4	4e-4	5e-4	6e-4	7e-4	8e-4	9e-4	10e-4
Accuracy [%]	99.0	99.1	99.0	99.2	98.8	99.1	99.2	99.0	**99.3**
Batch Size	100								
Exp. Decay Learning Rate	2e-4	3e-4	4e-4	5e-4	6e-4	7e-4	8e-4	9e-4	10e-4
Accuracy [%]	99.1	**99.2**	99.1	98.9	99.1	99.0	99.1	**99.2**	**99.2**

Based on the attained results, it can be inferred that the optimal learning rate for a batch size of 150 is 10e-4, resulting in an accuracy of 99.3%. Conversely, for a batch size of 100, accuracies of 99.2% were achieved with learning rates of 3e-4, 9e-4, or 10e-4. For simplicity purposes, 10e-4 will be the learning rate considered for both configurations on further examinations.

Lastly, an attempt is made to enhance the model's performance by adjusting the dropout, which was previously set at 0.5. Consequently, a series of tests are conducted with varying dropout values.

Table 4.5. - Accuracy values.

Batch Size	150									
Exp. Decay Learning Rate	10e-4									
Dropout	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Accuracy [%]	99.2	99.1	99.2	99.1	99.2	**99.3**	99.1	99.0	99.0	99.0
Batch Size	100									
Exp. Decay Learning Rate	10e-4									
Dropout	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Accuracy [%]	**99.2**	**99.2**	99.1	99.1	99.1	**99.2**	99.1	**99.2**	99.1	99.0

Based on the results obtained, no discernible improvement was observed. Therefore, the primary configuration and the corresponding achieved accuracies are summarized in the following table.

Table 4.6. - Accuracy values.

Batch Size	150
Exp. Decay Learning Rate	10e-4
Dropout	0.5
Accuracy [%]	99.3

5. EXERCISE 4

In the concluding exercise, the objective is to investigate the impact of the quantity of GPUs on the duration of model training. Accordingly, in accordance with the task's recommendation, the optimal model, as presented in Table 4.6, will be executed using 1, 2, and 4 GPUs on P9 Cluster, and the corresponding time measurements will be recorded for the purpose of drawing conclusions.

Consequently, the obtained outcomes are hereby presented in Table 5.1.

Table 5.1. - Training time.

Number of GPUs	1	2	4
Training Time [s]	19.254	18.463	18.234

Based on the obtained results, it is evident that the duration of training remains roughly steady independently of the increment in the number of GPUs employed during code execution. However, it is important to acknowledge that these findings cannot be universally applied to all types of deep learning architectures. Specifically, the outcome is predominantly influenced by the complexity of the architecture. Consequently, for exceedingly simplistic architectures, no discernible disparity is

expected, whereas the greater the complexity, the more pronounced the quantitative time enhancement becomes.

6. CODE

The code developed is organized into 5 files:

- **exercise1_1stpart.py**: contains exercise 1, where the Gradient Descent Optimizer with different learning rates is tested.
- **exercise1_2ndpart.py**: contains exercise 1, where different Optimizers are tested for 0.01 learning rate.
- **exercise2.py**: contains exercise 2, where different Optimizers with a set of learning rate values are tested.
- **exercise3.py**: contains exercise 3, where it is intended to improve the performance of the model.
- **exercise4.py**: contains exercice 4, where it is intended to understand the influence of the number of GPUs used.

7. CONCLUSION

In conclusion, the analysis conducted within this report has unveiled several pivotal findings concerning the optimization and training of deep learning models.

First and foremost, when examining the influence of different learning rate values on the Gradient Descent Optimizer, it was observed that a lower learning rate contributes to slower convergence but facilitates more precise weight adjustments. However, considering the relative simplicity of the problem under study, an optimal learning rate of 0.01 was identified.

Secondly, upon comparing various optimizers with a fixed learning rate, it became evident that both the Momentum and Nesterov Accelerated optimizers exhibit superior convergence, despite their graphical oscillation patterns.

In the subsequent analysis, it was consistently determined that the Adam Optimizer consistently produced the most favorable outcomes in terms of loss and accuracy. Specifically, learning rates of 0.005 and 0.01 were found to yield particularly favorable results when employed in conjunction with the Adam Optimizer.

In an endeavor to enhance the model's performance, modifications were made to the batch composition methodology. By randomizing batch sizes and incorporating an exponential decay mechanism for the learning rate, significant enhancements were achieved.

Lastly, the impact of GPU utilization on training duration was investigated. Since the complexity of the model was not significant the training time exhibited did not decrease significantly with an increment on the number of GPUs employed, specifically for the intricate architecture analyzed.

To summarize, this report underscores the utmost significance of meticulously selecting optimization techniques, learning rates, batch sizes, and GPU utilization to effectively optimize the training process and attain superior outcomes in deep learning tasks.