An Exploratory Study of Deep Learning for Predicting Computational Tasks Behavior in HPC Systems

Alexandre H.L. Porto
National Laboratory for Scientific Computing
Petrópolis, Brazil
xandao@lncc.br

Micaella Coelho
National Laboratory for Scientific Computing
Petrópolis, Brazil
micaella@lncc.br

Kary Ocaña
National Laboratory for Scientific Computing
Petrópolis, Brazil
karyann@lncc.br

Carla Osthoff
National Laboratory for Scientific Computing
Petrópolis, Brazil
osthoff@lncc.br

Francieli Boito
University of Bordeaux
CNRS, Bordeaux INP, INRIA, LaBRI
Talence, France
francieli.zanon-boito@u-bordeaux.fr

Douglas O. Cardoso
Smart Cities Research Center
Polytechnic Institute of Tomar
Tomar, Portugal
douglas.cardoso@ipt.pt

***Abstract*—The scientific gateway BioinfoPortal for bioinformatics applications is hosted in the National Laboratory for Scientific Computing (LNCC) and is coupled to the Santos Dumont (SDumont) supercomputer environment. BioinfoPortal offers a catalog of bioinformatics software that benefits from the parallel and distributed architecture offered by LNCC. Task submissions consume SDumont nodes shared by other users of the supercomputer; thus, it is important they use the best configuration, which is defined as the best choice of the number of threads/nodes to be allocated for every task submission. This article presents an analysis using neural networks to estimate the computational time required to execute bioinformatics software in several scenarios using a pre-configured number of nodes and threads. Our goal is to demonstrate the performance behavior of software such as RAxML in Bioinfoportal, and which computational scenario can be chosen to efficiently execute software in SDumont. Results support that the neural networks are adequate to predict the variable elapsed time, Elapsed, to evaluate the relationships between input parameters, number of bootstraps (RAxML), number of threads, and number of nodes, and to identify the fastest configuration. The goal is to make BioinfoPortal a smart, efficient, and green gateway. In future studies, we propose to study more variables and predictors as well as other bioinformatics software in BioinfoPortal.**

***Index Terms*—neural networks, phylogenetic analysis, extra trees, performance prediction, performance modeling**

Douglas O. Cardoso acknowledges the financial support by the Foundation for Science and Technology (Fundação para a Ciência e a Tecnologia, FCT) through grant UIDB/05567/2020, and by the European Social Fund and programs Centro 2020 and Portugal 2020 through project CENTRO-04-3559-FSE-000158.

I. Introduction

The BioinfoPortal [1] is a bioinformatics gateway hosted in the National Laboratory for Scientific Computing (LNCC) and coupled to the computational environment of the Santos Dumont (SDumont) supercomputer. BioinfoPortal was developed under the architecture of the CSGrid middleware [24] and is managed by the National High-Performance Computing System (SINAPAD/LNCC). The gateway supports bioinformatics software and workflows executions, dependencies, and libraries, which are allocated in the SDumont environment. This way, the waiting time for a gateway submission depends on the performance execution of the tasks on the supercomputer. RAxML (Randomized Accelerated Maximum Likelihood) is a bioinformatics software [2] on BioinfoPortal which aims to generate the phylogenetic trees from a dataset of biological sequences. Each RAxML submission in BioinfoPortal generates a task to be performed on SDumont. RAxML is an application widely used for research in phylogeny to implement genetic comparison through "maximum likelihood" algorithms using complex and effective probabilistic models, which generate a high computational cost in terms of RAM memory.

The BioinfoPortal faces a challenge regarding the efficient use of SDumont resources, since each application requires a different set of parameters (number of nodes and processes) in order to achieve its best performance. The portal currently configures the same default parameters for all applications; therefore, to improve its performance and promote good, efficient, usage of SDumont, it is necessary to select the

979-8-3503-8160-3/23/$31.00 ©2023 IEEE
DOI 10.1109/SBAC-PADW60351.2023.00012

best set of parameters to be used in the execution of each application. There are different ways to deal with this situation, such as testing all possible application parameters. Another alternative would be to use machine learning (ML), which would allow obtaining the necessary information through a non-exhaustive set of tests and would be easily applicable to new applications on the portal.

Machine learning would allow the use of a model that helps choose the best set of parameters for each BioinfoPortal application, when it must be executed on SDumont. That would be possible after it has been trained with data obtained from real submissions made to this portal application. This can be done through supervised learning, in which predictive models can be trained from historical data from the behavior of executions of a set of non-exhaustive tests of an application. In this article, we propose to use neural networks to help discovering the ideal parameters for the execution of the RAxML application, available on the BioinfoPortal portal, as an alternative to the extra trees used in previous studies [11], [15], [14]. Neural networks are usually recommended more for situations with large data sets, because small data sets, when used to train the network, can cause it to overfit to that data. However, studies have shown that if well designed, neural networks can perform well for a small data sets as well [3]–[5].

This article, as well as the complete work, refers to a particular run of RAxML. As in previous studies [14], [15], [16], the objective is to use neural networks to determine the most effective parameters for submitting the RAxML application on the Santos Dumont supercomputer for efficient execution. Information to construct neural network models was returned from variables "number of computational nodes and number of threads per computational node" and the input parameters "bootstrap and input file size", the last ones supplied by the user. The secondary objective of this article is to compare the effectiveness of neural networks with Deep Learning algorithms for determining the submission parameters of the RAxML application for the Bioinformatics portal.

The structure of this paper is outlined as follows. Section II provides the basic concepts we need to know to understand the neural network proposed by the article, in addition to exploring previous studies based on extra trees. Section III describes the methodology used to define the structure of the proposed neural network for an undefined number of neurons and layers, to then show how to choose the network with the best number of neurons and layers for the proposal of our article. Section IV evaluates the best neural network defined by the previous section, comparing its prediction performance with that of the extra trees used in previous studies, while Section V offers concluding remarks and insights into possible future research directions.

II. Research Background

To be able to choose the optimal submission values for the number of nodes and the number of threads, according to the input parameters bootstrap and file size, associated with a RAxML execution submitted to the BioinfoPortal portal, we will train a neural network to predict the "Elapsed" metric, obtained by the command *sacct* of the SLURM task management system used by SDumont, referring to a certain execution of the RAxML. In our previous studies [14], [14], [16], we used extra trees to predict the value of this parameter in this context. Therefore, in this work, we compare the neural network-based solution to that one with extra tress.

The information given by the Elapsed variable was chosen because we aimed, for each execution of the RAxML program generated from a submission made to the BioinfoPortal portal, to choose the best parameters for the number of nodes and threads, which minimize the execution time, but without compromising the memory space used in the execution of the RAxML program. In other words, we want to satisfy a compromise between the time spent and the memory consumption, to avoid the program being too fast, but with significant memory consumption, or that the program is very slow, but with negligible memory consumption. Therefore, our objective is to train the network to predict the Elapsed variable so that the prediction of this parameter, given the number of nodes, the number of threads, and the bootstrap, is as close as possible to the value that the command *sacct* would obtain if a real RAxML execution were done using the same number of nodes, threads, and bootstrap.

A. Extra Trees

In previous studies [15], [14] and [16], the supervised machine learning method was adopted to predict the behavior of the RAxML application. This involved formulating individually supervised tasks for each output parameter, using node numbers, *threads*, and *bootstraps* as input parameters. A binary classification task was applied, wherein the values of the output parameters were categorized as either below or above the sample median. In other words, this approach utilized the median as the cutoff point to divide the data into two categories. The use of the median aimed to avoid problems related to data imbalance, which can arise in both classification problems [17] and regression problems [18]. For the binary classification analysis, cross-validation was used to compare the accuracy of the various classification tasks, aiming to evaluate the model's performance and identify the most influential input parameter in predicting the output parameters.

Next, the supervised regression task was considered. In this scenario, the emphasis is on numerical values of output variables rather than class labels. The regression technique allows for a more detailed analysis of the values, providing a numerical estimate of the variable of interest. For the regression analysis, cross-validation was used to compare the mean absolute error, which involved a comprehensive assessment of the model's performance. By evaluating the classification and regression models, we learned how well they performed on different aspects of the data. This allowed us to assess the ability to classify correctly and the accuracy of numerical estimates.

In conducting these supervised learning tasks, we used the *Extra Trees Classifier* and *Extra Trees Regressor* [19] models, employing the default settings provided by the Scikit-Learn library [20]. These models employ the same algorithm, generating many random decision trees and combining the results to obtain the final prediction. The main difference between the models lies in the type of problem they were developed to solve. The choice of these models is due to the advantage of not requiring complex configurations to achieve competitive performance compared to similar models. Furthermore, they provide an estimate, during the training itself, of the degree of influence of each input parameter on the prediction of the output parameter. This information is valuable for assessing the importance of the input parameters in predicting the output parameters.

B. Neural networks

Neural networks are one of the deep learning techniques used in artificial intelligence. The methodology of the technique is inspired by how the human brain works [6]. In this technique, several structures, called neurons or perceptions, are distributed in several interconnected layers, which aim to simulate how neurons connect in the human brain. The inner layers of the network are called hidden layers because we do not have direct access to the results of the computations generated by its neurons. The last layer is responsible for generating the results we want to predict, with a neuron for each result that needs to be predicted by the network.

To represent the neural network, we use a weighted graph built according to the existing layers in the network. As only the first-layer nodes are not associated with neurons, from now on, in the text, to simplify the notation, we will refer to the nodes of the graph associated with neurons simply as neurons. Considering the network layers from left to right, the first layer is the input layer, and there is a node in this layer for each input parameter used when training the network. The intermediate layers are represented by the neurons responsible for most of the processing that occurs when the neural network is trained. Finally, the last layer, the output layer, is responsible for the final calculation of the results to be predicted by the network, with a different neuron in the layer for each result predicted by the network. As the network processing starts from the second layer, an edge connects each neuron of the second layer to each of the nodes associated with the input parameters. For all layers from the third one, there is, for each neuron in the layer, an edge connecting it to all the neurons in the immediately previous layer. Note that there are no edges between neurons in the same layer. Each edge of the graph has a weight, which will be adjusted as the neural network is trained, with the objective that, after training, the weights of the edges are the most adequate for the neurons to generate the best possible predictions for the network outputs from the given inputs to the network.

Processing in the network proceeds from left to right, starting with the input data used to train the network, layer by layer, to the output layer, where the results predicted by the network are generated. As each neuron of a layer, except for the second, is connected with the neurons of the previous layer, the processing of this layer can only start after all the neurons of the previous layer have finished their processing. For each neuron t of a layer, its processing is quite simple, as it calculates the result of the following function $y = w_1x_1 + w_2x_2 + ... + w_nx_n$, where n is the number of neurons in the previous layer or the number of inputs in case t is in the second layer, x_i is the value generated by neuron i from the previous layer or input parameter x_i, and w_i is the weight of the edge connecting neuron t to neuron i or input parameter x_i. A function f, called activation function, is additionally applied to the value calculated by each neuron, that is, the value actually calculated by the neuron is $f(y)$. After neuron t computes its value, if the layer to which t belongs is not the last one, the result of t is passed to all neurons in the next layer. If the layer is the last one, then the result is the value predicted by the network for one of the output parameters. Figure 1 illustrates a simple neural network with tree input parameters (I_1, I_2 and I_3), two hidden layers, each with four neurons, and the output layer, with a result, O_1, to be predicted by the network.

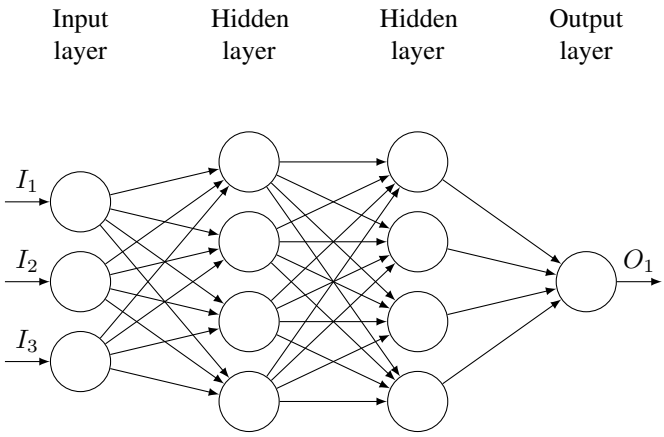

Fig. 1. Example of a neural network.

When we use neural networks, they are trained for several steps. Each step is called an epoch, and at each epoch, a given algorithm called an optimizer, changes the weights of the edges of the neural network according to the score used to evaluate the predicted results for the input data used when training. A loss score is generated for each epoch, which measures the model's prediction quality after that epoch for the input data used when training the network. If validation data is being used when training the network, then, in addition to the loss value, a score, called the val_loss, is generated, which measures the quality of the network prediction at that epoch for the validation data.

III. Methodology

As we observed in the previous section, a neural network is composed of several layers, one of which is the input, a given number of hidden layers, and an output layer, responsible for generating the values provided by the network. In this section, we will describe, in the following subsections, how we will build networks to predict the Elapsed variable from data obtained by real RAxML executions in SDumont.

A. The MAE score

When training each neural network for each combination of several neurons and layers described in the previous paragraph, and when comparing the prediction performance of the neural networks with the extra trees, we will use the MAE (Mean Absolute Error) [23] score. Consider that an artificial intelligence model predicted the outputs $x_1, x_2, \ldots, x_n$ for a given set of input data. If the actual outputs, that is, those obtained by actual runs on these same input data, are $y_1, y_2, \ldots, y_n$, with each y_i being the actual value of the predicted x_i value, then the MAE score for the predicted values will be the average of the modulus of differences between each predicted value and the actual value it corresponds to, that is, $\left(\sum_{1 \leq i \leq n} |x_i - y_i|\right) / n$.

B. Getting the best numbers of neurons and hidden layers

To choose the best network to compare to the performance of the extra trees, we made several network configurations for the Elapsed variable that we are going to use in the performance evaluation tests in this article, obtained by varying the number of neurons in each layer in the set $\{1, 4, 16, 64, 256, 1024\}$, and the number of hidden layers varying from 1 to 10, giving a total of 60 different combinations of number of hidden layers and number of neurons per hidden layer. In preliminary tests, we tried to use the same configuration as in article [3], a neural network with 100 neurons per layer and ten hidden layers, because this article, to our knowledge, is the most referenced when neural networks are used with small datasets, but the results we obtained with it were not promising. To discover the most suitable network, with the best number of neurons per layer and hidden layers, we evaluated the number of layers ranging from 1 to 10 used in [3]. As for the number of neurons, we decided to evaluate smaller and larger numbers of neurons than the 100 used in [3], and, to reduce the neuron number space considered, we decided to use powers of 4 lesser and greater than 100.

In Figure 2, we show the heatmap graph for each combination of several neurons and the number of layers for the neural network used to predict the Elapsed variable. The values given in each heatmap entry are the averages for 40 tests with the number of neurons and layers associated with the entry, of the value obtained by training, using the MAE score, after 200 epochs, the same number used in [3]. We decided to run 40 tests and calculate the average because otherwise, in one of the 60 configurations, only one neural network would be modeled, for which we randomly chose 90% data for training and 10% for validation. Therefore, we tried to statistically evaluate the values for each possible combination of several neurons and layers with the 40 tests. It is important to notice that, for the MAE score, the best value is 0 and that the closer to 0, the better the prediction of the network according to the metric. We can see that, for the Elapsed variable, the best configuration is a network with two layers and 256 neurons in each hidden layer.

C. Evaluating the best result

In Figure 3, we show the graph with average values for the 40 tests performed, to the best-chosen combination, 256 neurons, and two hidden layers for all 200 epochs made while training the network. As we can see from the figure, the curves of the loss, related to the data used by the training, and the val_loss, related to the validation data, are very close, with a very small difference between the values. Furthermore, the decreases in loss and val_loss values stabilize after a few epochs. Previous evidence and these curves support that the architecture with 256 neurons and two hidden layers was at least as good as any of its alternatives and that additional training for more epochs could cause an overfitting of the network to the data used to train it.

D. Defining how the best neural network is configured

The purpose of the proposed neural network for the variable Elapsed is, based on the predictions of the possible values for this variable, to help choose the best values for the parameters number of nodes and number of threads. The network will consist of the following layers:

- Since the three parameters, "Node", "Thread" and "Bootstrap", are considered in each execution of the RAxML program, the input layer will be composed of three nodes. Due to the input parameters having very different ranges of values, normalized versions of these parameters were used, to ensure that all parameters have equal contribution in the training of the network. The normalization performed was min-max, which maps, for each parameter, its set of possible values in the interval $[0, 1]$, that is, if the parameter value is x, the minimum value of the parameter is min and the maximum value of the parameter is max, then the new input parameter will be $(x - min)/(max - min)$.

- For the hidden layers, we will use the composition determined by the best heatmap value given in Figure 2. Therefore, our network will consist of 2 hidden layers, each consisting of 256 neurons. The He-initialization method [7] will be used to initialize the edge weights of the neural network. The Adam optimizer [8], with a learning rate of 0.01, will be used when optimizing the weights of the network, and training will occur for 200 epochs, as done in the article [3]. The function used by the Adam optimizer, when training the network and adapting the weights of the edges, will be the "mae" based on the mean absolute statistical (MAE) error. For

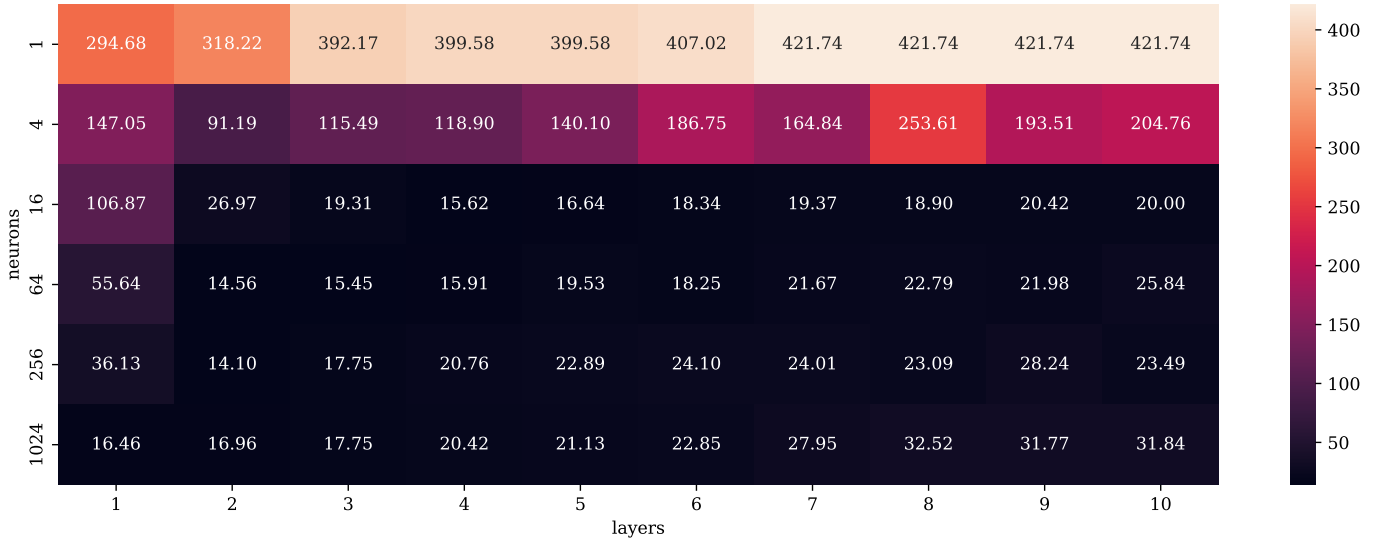

Fig. 2. Heatmap for the Elapsed variable with MAE metric values for all combinations of numbers of neurons and layers described in the text. Colors vary in intensity according to the MAE value, being darker for smaller values, as shown by the color scale on the map's right side. The best combination, with two hidden layers and 256 neurons per layer, is associated with the map entry with the lowest MAE value of 14.10.

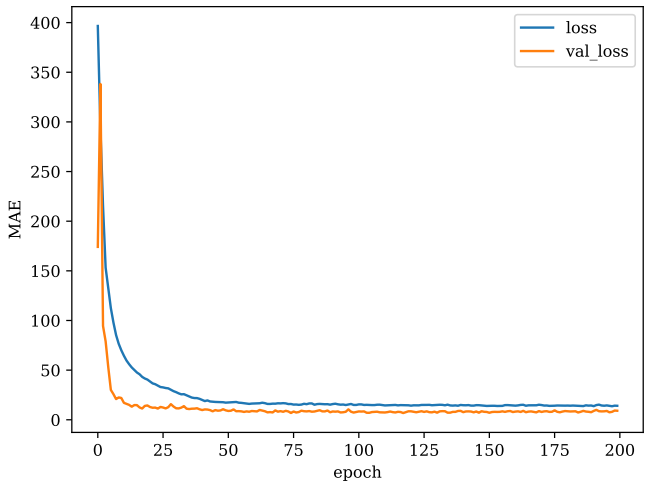

Fig. 3. Loss and val_loss values calculated at each of the 200 epochs when training the model with 256 neurons per layer and two hidden layers.

the activation function of each neuron, we decided to use the "relu" function most used in studies based on neural networks (for this function, $f(y) = \max(\{0, y\})$, instead of the "elu" function [9] used in [3], because our results were better with this function.

- Finally, the last layer will be composed of a neuron that will be responsible for computing the value predicted by the network, that is, the value of variable Elapsed.

IV. Experimental results

The neural networks were implemented in the Python language using the Keras library [10] and the Sequential class of this library, defined to facilitate the creation of very complex neural networks. The class object was configured to have an input layer for the three parameters "number of nodes", "threads" and "bootstrap", and two hidden layers defined with 256 neurons and the initialization function of the weights He-initialization. Adam was defined as the optimizer with a learning rate of 0.01 when compiling the network. Additionally, we choose the MAE as the evaluation function used by the optimizer, called the loss function, and defined by the "loss" option.

When the network was trained using a given dataset, 90% of the training data was used to effectively train the network, while 10% of the training data was used to validate the network while it was being trained. This division evaluated the network in each of the 200 epochs for which it was trained by the optimizer Adam, who, as we saw, uses the MAE scoring function described above. The set of input data available to train the network, obtained from real RAxML executions in SDumont, was divided equally, in two parts, 100 times, using the RepeatedKFold function of the model_selection module of the sklearn library [20], as done in previous studies based on extra trees [14], [15], [16]. For each of the 100 splits into two parts, the RepeatedKFold function returns two sets of parts, one where the first part is used to train the network and the second part is used as test set to evaluate the network's prediction accuracy, and another where the roles of the parts are swapped, i.e., the first part is now used for evaluating accuracy while the second part is used for training.

To train the neural networks for the Elapsed parameter, tests were carried out in SDumont, using the number of nodes in the

set $\{1, 2, 4\}$, the number of threads in the set $\{2, 4, 8, 12, 24\}$, and the bootstrap of the set $\{100, 500, 1000, 1500, 2000\}$. The amino acid sequences used were from the aminoacido.phylip file, composed of amino acids associated with ten species for which we wanted to build the best phylogenetic tree describing the kinship of these species. To avoid influences of runtime variations due to the shared use of SDumont, each combination of node and thread numbers and bootstrap parameters was executed five times. Therefore, the total number of tests executed should have been $3 \times 5 \times 5 \times 5 = 375$, but the total was a little less, 370 tests, because five tests were aborted due to execution problems in SDumont. Using the test set, each network was trained as described in the previous section.

To evaluate the efficiency, we will compare the neural network of the variable Elapsed with a previously created model [11], [14], [15], [16] for each of the parameters, using the extra trees based on random decision trees [12] [13]. To compare the accuracy of the neural network and extra tree prediction, let's use the MAE score used when training the neural network. For every 200 tests described above, both the extra trees and the neural networks for the Elapsed variable will be generated using the data defined for training and then will be evaluated using the data defined for prediction, being that for each predicted result, we will calculate its MAE score with the given result in the data used for prediction.

In Figure 4, we show the boxplot plot of the MAE score for all 200 predictions made by the extra trees and neural networks with two hidden layers with 256 neurons. The figure shows that the neural network performs worse than the extra tree since, for the MAE score, the best values are always the smallest, with 0 being the ideal value (in this case, the predicted values would be exactly equal to the real values). Since neural networks typically excel on complex problems with a relatively large set of variables as well as more abundant training data, it could be expected that the results of the neural networks were better compared to those of the extra trees in a scenario where more parameters and performance metrics of the computational tasks are considered.

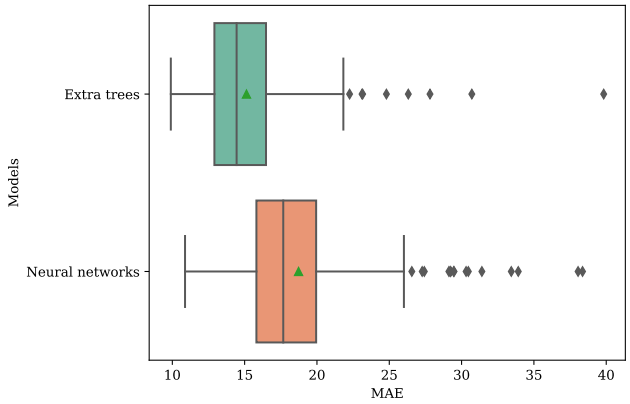

Fig. 4. MAE scores for predictions from extra trees and neural networks. In the graph, the MAE scores that are numerically farthest from the majority of the 200 scores are shown outside the range delimited in each boxplot by diamonds. Considering the scores that are within the range, the range bounds define the smallest and largest value of the scores within the range. Still considering the scores within the range, for each filled rectangle, its boundaries represent the first and third quartiles of these scores, the inner line represents the mean of these scores, and the green triangles represent the medians of these scores.

V. Conclusion

From the results of the previous section, we can conclude that neural networks are also adequate to predict the Elapsed variable considered in the article, and we believe that it is justifiable not only to perform new training with more data but also to evaluate the prediction performance of neural networks in comparison with extra trees for each of the other five variables MaxVMSize, AveVMSize, MaxRSS, AveCPU and CPUTime described in [11]. It is noteworthy that, although we need to rebuild the models generated by the extra trees used in previous studies, for a neural network, we can keep training it without having to rebuild it from scratch.

A possible future study is, as done in [11] for the extra trees, to study the accuracy, for neural network predictions, of each of the input parameters, Node, Threads and bootstrap, for the Elapsed variable considered in this article, and for the neural networks to be trained for the remaining variables MaxVMSize, AveVMSize, MaxRSS, AveCPU and CPUTime. As these variables can be grouped into two groups, time-related variables (AveCPU, CPUTime, and Elapsed) and memory usage-related variables (MaxVMSize, AveVMSize and MaxRSS), we could also develop a neural network for each one of the two groups, which would predict, from the Node, Threads, and bootstrap input parameters, values for all variables in the group.

In this article, we consider the construction of a neural network for an output variable, Elapsed, which measures the complete execution time of the RAxML program. Of all six variables considered in [11], the variables (AveCPU, CPUTime, and Elapsed) are related to execution time, and the variables MaxVMSize, AveVMSize, and MaxRSS are related to memory expenditure. However, there are applications that, instead of spending most of their time computing and demanding a lot of CPU time, i.e., CPU-bound applications, spend most of their runtime accessing files, i.e., IO-bound applications. Therefore, a possible future study would be to consider the output variables of the sacct SLURM command related to the I/O operations performed by an application, such for example, the variables AveDiskRead, AveDiskWrite, MaxDiskRead, and MaxDiskWrite.

To evaluate the possibility of adapting neural networks to other programs, we propose, as future studies, to define new neural networks to predict the variable Elapsed and the other five defined in [11], for another application of the BioinfoPortal portal, bowtie2 [21], used to perform various sequence alignments. In the case of this application, which is just multitasking, we will evaluate only the number of threads since the number of nodes is always equal to 1, and as an application-dependent parameter, we will use the size of the file with the sequences to be aligned. Later, we intend to make

neural networks that adapt to any application, using two sets of inputs: the ones dependent on the execution in SDumont and those dependent on the application.

ACKNOWLEDGMENT

We would like to thank CNPq for providing the resources for the project, INRIA and its collaboration with the project, and the Santos Dumont supercomputer (SDumont) support team.

REFERENCES

[1] K. Ocaña, M. Galheigo, C. Osthoff, L. Gadelha, F. Porto, A.T.A. Gomes, D. Oliveira and A. Vasconcelos, "BioinfoPortal: A scientific gateway for integrating bioinformatics applications on the Brazilian national high-performance computing network", Future Generation Computer Systems, Vol. 107, 2020.

[2] A. Stamatakis, "Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies," Bioinformatics, vol. 30(9), pp. 1312-–1313, 2014.

[3] M. Olson, A. J. Wyner and R. Berk, "Modern neural networks generalize on small data sets," NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 3623-–3632, December 2018.

[4] T. Shaikhina and N. A. Khovanova, "Handling limited datasets with neural networks in medical applications: A small-data approach," Artificial Intelligence in Medicine, vol. 75, pp. 51–63, January 2017.

[5] A. Pasini, "Artificial neural networks for small dataset analysis," J. Thorac. Dis., vol. 7(5), pp. 953—960, May 2015.

[6] W.S. Mcculloch and W. Pitts, "A Logical Calculation of the Ideas Immanent in Neuvous Activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133, 1943.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," IEEE International Conference on Computer Vision (ICCV), pp. 1026—1034, 2015.

[8] D.P. Kingma and J.L. Ba, "Adam: Amethod for stochastic optimization," In Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[9] D.A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," arXiv preprint arXiv:1511.07289, 2015.

[10] F. Chollet, "Deep learning with python," Manning Publications Co., 2017.

[11] M. Coelho, C. Osthoff and K. Ocaña, "Avaliação do RAxML no Supercomputador Santos Dumont," Anais Estendidos do XI Simpósio Brasileiro de Bioinformática, 2018.

[12] L. Breiman, "Random forests," Machine Learning, vol. 45, pp. 5-–32, 2001.

[13] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," Machine Learning, vol. 63(1), pp. 3-–42, April 2006.

[14] Coelho, M. et al. Development of a Machine Learning Framework to Support Efficient Scientific Gateways. In: Latin America High Performance Computing Conference (CARLA 2022), Porto Alegre, 2022.

[15] COELHO, Micaella et al. Desenvolvimento de um Framework de Aprendizado de Máquina no Apoio a Gateways Científicos Verdes, Inteligentes e Eficientes: BioinfoPortal como Caso de Estudo Brasileiro. In: Anais do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho, 2022, Florianópolis. Sociedade Brasileira de Computação, 2022. p. 205-216.

[16] Coelho, M. et al. Um estudo sobre Aprendizado de Máquina no Apoio a Gateways Científicos Verdes, Inteligentes e Eficientes. IN: XXIII Encontro da Rede de Estudos Ambientais de Países de Língua Portuguesa - REALP, Instituto Politécnico de Tomar, 2022.

[17] Johnson, Justin M. and Khoshgoftaar, Taghi M. Survey on deep learning with class imbalance. In: Journal of Big Data, vol. 6, no. 1, p. 27, Mar. 2019.

[18] Ribeiro, Rita P. and Moniz, Nuno Imbalanced regression and extreme value prediction. In: Machine Learning, vol. 109, no. 9, pp. 1803–1835, Sep. 2020.

[19] Geurts, Pierre et al. Extremely randomized trees. In: Machine Learning, vol. 63, no. 1, pp. 3–42, Apr. 2006.

[20] Pedregosa, Fabian et al. Scikit-learn: Machine Learning in Python. In: Journal of Machine Learning Research, vol. 12, no. 85, pp. 2825–2830, 2011.

[21] B. Langmead and S.L. Salzberg, "Fast gapped-read alignment with Bowtie 2", Nature Methods, Vol. 9, pp. 357—359, 2012.

[22] A.B. Yoo, M.A. Jette and M. Grondona, "SLURM: Simple Linux Utility for Resource Management", Workshop on Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, pp. 44–60, 2003.

[23] C.J. Willmott*, K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance", Climate Research, vol. 30, pp. 79—82, 2005.

[24] A. T. A. Gomes, B. F. Bastos, V. Medeiros, V. M. Moreira, "Experiences of the brazilian national high-performance computing network on the rapid prototyping of science gateways", Concurrency and Computation: Practice and Experience, Vol. 27, No. 2, pp. 271—289, 2015.