COMPDYN 2021
8th ECCOMAS Thematic Conference on
Computational Methods in Structural Dynamics and Earthquake Engineering
M. Papadrakakis, M. Fragiadakis (eds.)
Streamed from Athens, Greece, 28-30 June 2021

PERFORMANCE AND SCALABILITY OF DEEP LEARNING MODELS TRAINED ON A HYBRID SUPERCOMPUTER: APPLICATION IN THE PREDICTION OF THE SHEAR STRENGTH OF SLENDER RC BEAMS

Nikolaos Bakas[1], George Markou[2], Dimos C. Charmpis[3] and Kyriakos Hadjiyiannakou[4,1]

[1]Computation-based Science and Technology Research Center, The Cyprus Institute
20 Konstantinou Kavafi Street, 2121, Aglantzia Nicosia, Cyprus.
e-mail: n.bakas@cyi.ac.cy

[2] Structures Division, Civil Engineering Department, University of Pretoria
Hatfield Campus, 0028 Pretoria, South Africa.
e-mail: george.markou@up.ac.za

[3] Department of Civil and Environmental Engineering, University of Cyprus
75 Kallipoleos Str., P.O. Box 20537, 1678 Nicosia, Cyprus
e-mail: charmpis@ucy.ac.cy

[4] University of Cyprus, Department of Physics
e-mail: k.hadjiyiannakou@cyi.ac.cy

Keywords: High-Performance Computing, Deep Learning, Artificial Neural Network, Finite Elements, Nonlinear Analysis, Reinforced Concrete, Slender Beams.

Abstract. *Data-driven models employing artificial intelligence approaches have been increasingly utilized in structural analysis and design problems over the past two decades. The main applications involve the processing of datasets, which are gathered from experimentally derived records or obtained numerically, in order to develop closed-form formulae or numerical tools predicting quantities related to structural response and mechanical behaviour. Given that datasets are difficult to assemble due to the limited available information and the high cost entailed to enrich them, exhaustively processing the available data to produce the best possible prediction models is an essential task of particular interest. For specific applications, this exhaustive computing task involves large numbers of iterations performed to train detailed prediction models with large numbers of parameters. Despite the intense computational demands of such problems, limited research work exists on the scaling-up of the utilized algorithms on supercomputers. In this work, a distributed training and hyperparameter tuning algorithm is proposed for the modelling of the shear strength of slender beams without stirrups. The training dataset comprises results obtained from the detailed modelling and analysis of several beams with non-linear finite elements using the Reconan software. The results presented in this research work highlight the importance of optimally utilizing computational power for the solution of such problems. The developed computer code is available on GitHub.*

1 INTRODUCTION

Artificial intelligence techniques have emerged over the last decades as an effective and efficient tool to predict analysis outputs for computationally demanding engineering problems. Application areas requiring multiple analysis runs (design optimization, structural reliability, etc.) have largely benefited from various computational approaches eliminating the need for performing actual analyses by providing adequate estimations for the outputs of interest (e.g. [1, 2, 3, 4]).

The training process of an Artificial Intelligence model is itself a demanding task in terms of computational resources. Particularly, an Artificial Neural Network can comprise millions of parameters to train via iterative procedures, such as the stochastic gradient descent algorithm. These algorithms yield data structures that frequently cannot fit in the GPU (graphics processing unit) accelerators' RAM; hence parallelization is vital for the accomplishment of the training task, and obtaining accurate results. The optimization algorithms employed during the training process of a deep network can be parallelized by following either of two main routes. The first one is data parallelism [5, 6, 7], which foresees the splitting of the "batch of samples" (utilized in each iteration) into a number of smaller mini-batches, which are processed in parallel, depending on the number of available resources (GPUs). Alternatively, we may use model parallelism [8], by partitioning the deep learning model on distributed GPUs.

Despite the fast-pacing growth of deep learning, along with the vast need for computational resources, there are rather limited relevant engineering applications reported in the literature [9, 10, 11, 12, 13], none of which concerns reinforced concrete (RC) structures or slender beams. The purpose of this work is to investigate deep-learning algorithms for the prediction of the shear strength of RC beams. For the needs of this research work, detailed 3D finite element (FE) modelling of RC structures is adopted [14, 15, 16, 17, 18], in order to develop a large database of results on slender RC beams without stirrups. The numerically generated data set is then used to train models to predict the beams' shear strength. A new algorithm is designed and programmed, in order to be able to develop multiple input files (FE models) and efficiently analyze them through the use of the software Reconan FEA [19]. The newly developed algorithm (Reconan Multirun) and Reconan FEA were used to generate and analyse approximately 36,000 slender RC beams without stirrups; the obtained results were used for the training of the predictive models.

2 DATABASE ASSEMBLY

For each beam considered, 10 independent variables (input) and one dependent variable, which is the load corresponding to the ultimate strength (output), are varied. Particularly, the basic geometric variables are varied for each beam: the net span L (mm), the width b (mm), and the effective depth d (mm) of the beam. Apart from these variables that affect the FE meshing of the beams, strength and material related variables are also varied, and, particularly, f_c (uniaxial compressive strength of a cylindrical specimen in MPa), E_c (concrete Young modulus in MPa), f_t (concrete tensile strength ratio), β (remaining shear capacity strength factor), E_s (steel Young modulus in MPa), f_y (steel yielding stress in MPa), and ρ (tensile longitudinal reinforcement ratio).

Sampling for the 10 independent variables was performed uniformly as per Table 1, with the variables' coefficient of variation $c_v = \frac{\sigma}{m}$ (standard deviation over mean) being kept within the range 0.2 to 0.5 [20]. Hence, for each group of beams, different values for geometric and strength variables were generated, and the overall database is finally constituted of 35,849

Table 1: Statistical metrics of independent variables.

	L	d	b	f_c	E_c	f_t	β	E_s	f_y	ρ
Minimum	1500.0	260.0	200.0	20.0	25000.0	0.020	0.020	1.90×10^5	400.0	0.0010
Maximum	8700.0	1310.0	600.0	60.0	35000.0	0.100	0.050	2.10×10^5	600.0	0.0200
c_v	0.416	0.406	0.228	0.305	0.097	0.404	0.252	0.029	0.115	0.627

observations in total. Out of the total database population, 85% was used as a training set and the remaining 15% was used for testing.

3 BASELINE AND MACHINE LEARNING MODELS

The predictive modelling of the database was implemented by utilizing four machine learning (ML) methods, and in particular Linear Regression (LR), Non Linear - higher order Regression (NLR) [21], Random Forests (RF) [22] as implemented in [23], and Gradient Boosting [24] (GB) as implemented in [25]. The performance of each ML method varies, and the Mean Absolute Percentage Error (MAPE) [26] was used as a meaningful metric for engineering applications. The MAPE in the test set for the four methods was 10.843 (RF), 12.178 (GB), 22.036 (NLR), and 32.211 (LR). Random forests exhibit the lowest MAPE for the test set, while LR the highest, which was expected as it is the simplest (linear) model used. Nonlinear regression (NLR) exhibits higher error than RF and GB, however, the NLR model is useful due to the closed-form formula that it generates. The distribution of ML models' residuals exhibited a shape close to the Gaussian, for all methods except the linear regression, which was skewed. This is due to its inability to capture the non-linear behaviour of the variables included in the model, highlighting the need for more complex ML models.

4 DISTRIBUTED DEEP LEARNING

PyTorch was used herein which is an "imperative style, high-performance deep learning library" [27], distinguished for scientific as well as industrial projects, due to a straightforward yet efficient implementation of an automatic differentiation algorithm [28]. For multi-GPU training, the Horovod [29] was implemented, a library that has been developed at Uber. Particularly, by using Horovod, one may take a single-GPU training script and efficiently scale it to run across many GPUs in parallel. With MPI commands [30], each process is initialized and is assigned its MPI rank in a straightforward manner, which is achieved with fewer code changes compared to other approaches. Ultimately, Horovod scripts can run on a single-GPU, multiple-GPUs, or even multiple hosts without any further code changes. Algorithms on various experiments were tested for the needs of this research work on the Cyclone Supercomputer[1], utilizing PyTorch for computer vision as well as regression tasks[2], highlighting the efficiency of data parallelism, as well as the scaling-up capabilities compared with standard ML platforms, such as Kaggle and Google Colab.

4.1 Parallel Stochastic Gradient Descent

In Stochastic Gradient Descent, at each iteration, a random data-point i is selected, the Loss Function for this data-point is differentiated, and then the weights for the entire network are

[1] https://hpcf.cyi.ac.cy/

[2] https://github.com/CaSToRC-CyI/artificial-intelligence-hpc

updated. In mini-batch Stochastic Gradient Descent, i is the current "batch" of data, which is a subset of all dataset indices. The parallelization is performed at this point, by updating the weights for all batches in parallel. Hence, in Parallel Stochastic Gradient Descent, for each GPU (in parallel) a random data-point is selected, for which the gradient is computed and the weights are mixed. The average for the mixing of the weights can be implemented for this purpose, or other methods, such as ensembles, AdaSum, etc. Afterwards, the update of the weights for all GPUs takes place. In order to perform this operation in parallel, the utilization of MPI[3] framework is required, so as to gather the results among all GPUs, reduce, and broadcast them across all available GPUs again. This algorithmic procedure is described in Figure 1.

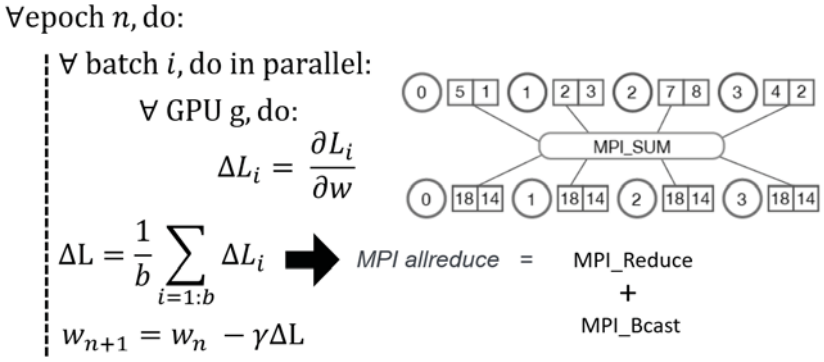

Figure 1: Parallel Stochastic Gradient Descent.

In general, it applies that, when using larger batch sizes the procedure becomes faster, and when adopting smaller batch sizes it becomes more accurate. In any case, the batch must fit into the relevant hardware memory, so the utilization of a larger number of nodes is required when a GPU cannot handle the batch size. Furthermore, in practice, the weights' mix causes some accuracy losses. However, all these aspects regarding the computational behaviour of the Parallel Stochastic Gradient Descent procedure are not always absolutely valid, as the loss function is a highly non-linear function of the Artificial Neural Network's (ANN's) weights and depends on the dataset's particular features, thus the computational procedure's behaviour cannot be predicted. Therefore, the training of the network has to be performed by investigating a variety of architectures. In order to effectively treat this issue, hyperparameter tuning is adopted in the present work, and, ultimately, the training of a very large network on Cyclone supercomputer is attempted.

4.2 Hyperparameter tuning

In order to investigate the effect of the network's hyperparameters, the Ray-Tune[4] module of PyTorch is implemented. Particularly, a number of ANNs are constructed by varying the batch size, the drop-out ratio, the number of Epochs and neurons, as well as the learning rate. Subsequently, the training of the networks is conducted. Accordingly, for each training example, the time for the training is recorded, while the final loss function and the ratio among the validation and train loss *Ratio V-T* are computed. The results obtained are presented in the Appendix. This

[3]https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/
[4]https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html

is a time-consuming procedure, therefore, it was found useful to identify an optimal drop-out region of ratios [31]. As depicted in Figure 2, it can be concluded that, for lower drop-out values, the corresponding values of the loss function are also lower. However, as will be explained below, the drop-out could not be zero.

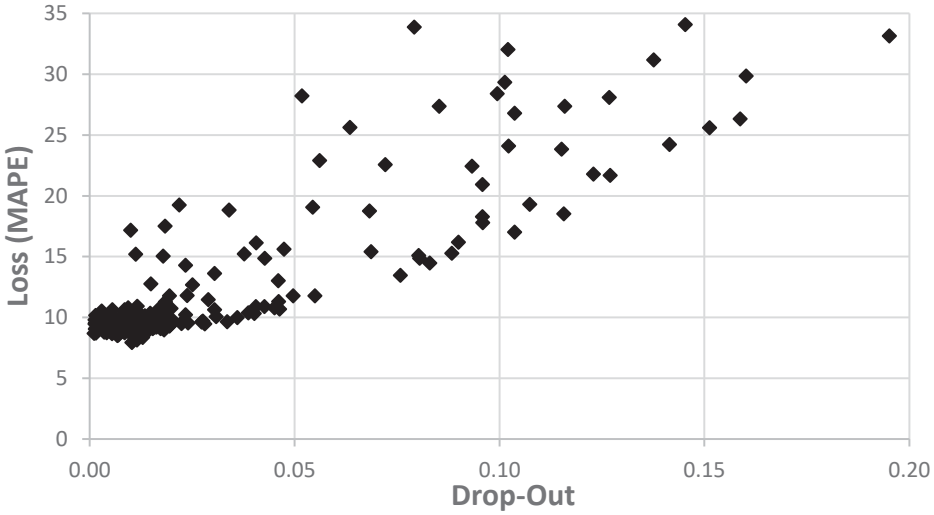

Figure 2: Ray-Tune results.

4.3 Combination of MPI & Horovod

During the training of an ANN, the loss function is automatically recorded, as loss is the core function that is differentiated to update the weights. In engineering applications, the Mean Squared Error (MSE) does not offer meaningful information, and the MAPE is being utilized as a more practical metric. Accordingly, as the best ANN's architecture is not known in advance, and given that multiple experiments must be performed, it is necessary to record the MAPE during training in real-time. However, as the data are split and reside at many GPUs, an operator should be implemented that gathers the individual predictions to obtain an aggregated metric for the dataset as a whole.

For this purpose, a mix of Horovod with MPI is proposed. In Figure 3, three code snippets are depicted describing the implementation of such an operation. The overall operation starts with the initialisation of vector $pred$ in the code at the upper right part of Figure 3, which is used later as a container of all results. The code at the upper left part imports the MPI module from mpi4py and gets the rank of each GPU. Afterwards, as shown in the code at the bottom-left part of Figure 3, after each step of the optimizer, a barrier is set to wait for all GPUs to finish with their numerical processes. Then, the responses y_train_pred of each GPU are stored in vector $pred\_i$ and, with the *comm.gather* command, all results are placed in $pred\_i$. Finally, an additional barrier is introduced to wait for all GPUs to finish and concatenate the results in vector $pred$ only for rank 0. At this stage, vector $pred$ holds the concatenated predictions and can be utilized to calculate the MAPE.

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
torch.manual_seed(0)
```

```
for e in tqdm(range(1, EPOCHS+1)):
    if rank  == 0:
        t1=time()
        pred = np.empty(0)
        predV = np.empty(0)
    else:
        pred = None
        predV = None
```

```
optimizer.step()
comm.Barrier()
pred_i = copy(y_train_pred[:,0])
pred_i = comm.gather(pred_i, root=0)
comm.Barrier()
if rank  == 0:
    for ii in range(len(pred_i)):
        pred = np.concatenate((pred,pred_i[ii].detach().cpu().numpy()), axis=0)
comm.Barrier()
```

Figure 3: Combination of MPI and Horovod.

4.4 Parallel training

Figure 4 demonstrates the learning curves for the model without using a drop-out and with a 10% drop-out. In the case of no drop-out (Figure 4, left), even though there is in general a decreasing tendency in MAPE with the number of epochs, a number of significant peaks are evident. This can be interpreted as an effect of the averaging of the weights in parallel, which is a source of error. However, with the utilisation of a 10% drop-out (Figure 4, right), smoother learning curves are achieved and the aforementioned phenomenon seems to be not activated. Furthermore, it should be stressed that the validation curve for 10% drop-out is lower than the training curve, which is very important for the generalisation of the results.

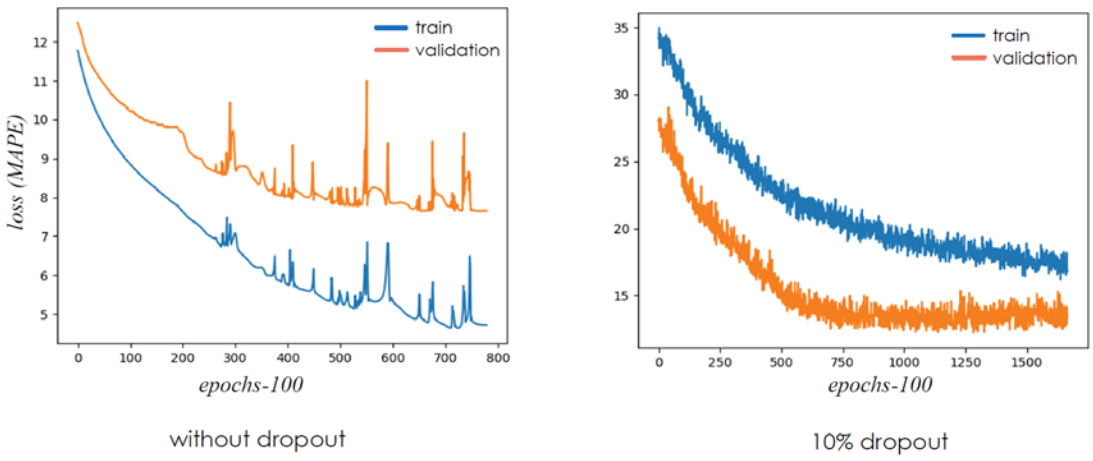

Figure 4: Drop-out effect on the loss function during training in parallel. The model with drop-out exhibits a validation loss history with lower values than the corresponding train curve.

It is noteworthy to state that a variety of experiments was performed to attain an optimal drop-out ratio. Figure 5 presents results for a drop-out equal to 0.01. As can be seen at the right part of Figure 5, when the validation curve is higher than the training curve, it practically stabilizes around a constant value. This is helpful during training, as over-training of the network will not cause over-fitting of the derived predictive model.

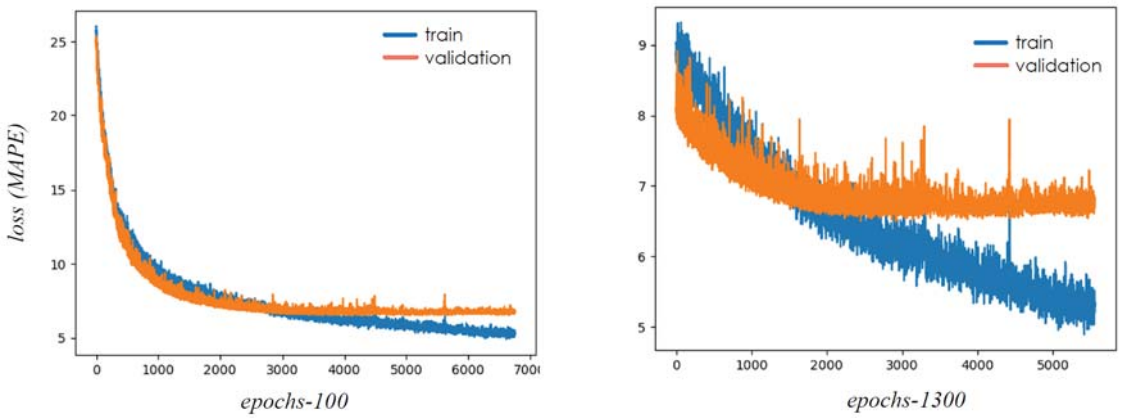

Figure 5: Results for low drop-out=0.01.

5 CONCLUSIONS

The numerical experiments performed in the present work comprised networks with large numbers of neurons, and thus, weights. For example, a network with 1000 neurons per layer and 10 layers corresponds to approximately 10 million weights to be optimised. As the most suitable network architecture is not known a-priori, several experiments need to be performed towards establishing the minimum possible error. Hence, computational power is vital for the best possible accuracy of the predictions.

It was also found that using PyTorch and Horovod as a functional solution for running deep-learning models on Cyclone Supercomputer is a very efficient approach. Furthermore, according to the parametric investigation performed in this research work, deep learning exhibited high accuracy in comparison to other ML methods. The best loss attained (MAPE) was 5.94%, while with Random Forests the corresponding loss was 10.7%, with Gradient Boosting 12.7% , Non-Linear Regression 24.0%, while Linear Regression had the worst numerical response with 32.3% loss.

The dataset under study comprised values with an increment of 10 kN and an average of 98.22 kN, thus an error $\frac{10/2}{98.22} = 5.091$ inevitably occurs on average. Henceforth, any improvement of the 5.94% loss attained would be trivial. This highlights the power of deep learning on one hand, however, the computational demand for training as well as optimising the architecture of such networks is very expensive. Distributed training made these operations possible herein, achieving both efficient and accurate results.

ACKNOWLEDGEMENTS

This work used resources from Cyclone Supercomputer[5] and was funded by the EuroCC project GA 951732. The use of computer resources under project Ispre519s1 of The Cyprus Institute is also acknowledged.

REFERENCES

[1] N. D. Lagaros, D. C. Charmpis, and M. Papadrakakis, “An adaptive neural network strategy for improving the computational performance of evolutionary structural

[5]https://hpcf.cyi.ac.cy/

optimization," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 30, pp. 3374–3393, 2005. [Online]. Available: https://doi.org/10.1016/j.cma.2004.12.023

[2] V. Sundar and M. D. Shields, "Surrogate-enhanced stochastic search algorithms to identify implicitly defined functions for reliability analysis," *Structural Safety*, vol. 62, pp. 1–11, 2016. [Online]. Available: https://doi.org/10.1016/j.strusafe.2016.05.001

[3] A. Roy, R. Manna, and S. Chakraborty, "Support vector regression based metamodeling for structural reliability analysis," *Probabilistic Engineering Mechanics*, vol. 55, pp. 78–89, 2019. [Online]. Available: https://doi.org/10.1016/j.probengmech.2018.11.001

[4] K. Singh and R. Kapania, "Alga: Active learning-based genetic algorithm for accelerating structural optimization," *AIAA Journal*, vol. 59, no. 1, pp. 330–344, 2021. [Online]. Available: https://doi.org/10.2514/1.J059240

[5] H. Li, A. Kadav, E. Kruus, and C. Ungureanu, "Malt: distributed data-parallelism for existing ml applications," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–16.

[6] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[7] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *arXiv preprint arXiv:1811.03600*, 2018.

[8] J. H. Park, G. Yun, M. Y. Chang, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, "Hetpipe: Enabling large dnn training on (whimpy) heterogeneous gpu clusters through integration of pipelined model parallelism and data parallelism," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 307–321.

[9] D. Abueidda, S. Koric, and N. Sobh, "Topology optimization of 2d structures with nonlinearities using deep learning," *Computers and Structures*, vol. 237, 2020.

[10] A. Gorshenin and V. Kuzmin, "Analysis of configurations of lstm networks for medium-term vector forecasting," *Informatika i ee Primeneniya*, vol. 14, no. 1, pp. 10–16, 2020.

[11] N. Do, A. Taberner, and B. Ruddy, "Design of a linear permanent magnet transverse flux motor for needle-free jet injection," 2019.

[12] R. Miller, B. Moore, H. Viswanathan, and G. Srinivasan, "Image analysis using convolutional neural networks for modeling 2d fracture propagation," vol. 2017-November, 2017, pp. 979–982.

[13] A. Oishi and G. Yagawa, "Computational mechanics enhanced by deep learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 327, pp. 327–351, 2017, advances in Computational Mechanics and Scientific Computation—the Cutting Edge. [Online]. Available: https://doi.org/10.1016/j.cma.2017.08.040

[14] G. Markou and M. Papadrakakis, "Accurate and computationally efficient 3d finite element modeling of rc structures," *Computers & Concrete*, vol. 12, 2013.

[15] C. Mourlas, M. Papadrakakis, and G. Markou, "A computationally efficient model for the cyclic behavior of reinforced concrete structural members," *Engineering Structures*, vol. 141, 2017.

[16] G. Markou, C. Mourlas, H. Bark, and M. Papadrakakis, "Simplified hymod non-linear simulations of a full-scale multistory retrofitted rc structure that undergoes multiple cyclic excitations – an infill rc wall retrofitting study," *Engineering Structures*, vol. 176, 2018.

[17] G. Markou, C. Mourlas, and M. Papadrakakis, "A hybrid finite element model (hymod) for the non-linear 3d cyclic simulation of rc structure," *International Journal of computational Methods*, vol. 16, 2019.

[18] G. Markou and W. Roeloffze, "Finite element modelling of plain and reinforced concrete specimens with the kotsovos and pavlovic material model, smeared crack approach and fine meshes," *International Journal of Damage Mechanics*, 2021.

[19] G. Markou, "v2.00," *Reconan, F.E.A. - User's Manual*, 2020.

[20] L. H. Koopmans, D. B. Owen, and J. I. Rosenblatt, "Confidence Intervals for the Coefficient of Variation for the Normal and Log Normal Distributions," *Biometrika*, 2006.

[21] N. P. Bakas, "NOESYS-AI Regression: A generic framework for predictive modeling and sensitivity analysis," 2018. [Online]. Available: https://noesys.net/regression-and-sensitivity-analysis/

[22] L. Breiman, "Random Forrests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] B. Sadeghi, "Decisiontree.jl," 2013. [Online]. Available: https://github.com/bensadeghi/DecisionTree.jl

[24] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[25] B. Xu and T. Chen, "Xgboost.jl," 2014. [Online]. Available: https://github.com/dmlc/XGBoost.jl

[26] T. Dimopoulos, H. Tyralis, N. P. Bakas, and D. Hadjimitsis, "Accuracy measurement of Random Forests and Linear Regression for mass appraisal models that estimate the prices of residential apartments in Nicosia, Cyprus," *Advances in Geosciences*, vol. 45, pp. 377–382, 2018.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[29] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[30] L. Clarke, I. Glendinning, and R. Hempel, "The mpi message passing interface standard," in *Programming environments for massively parallel distributed systems*. Springer, 1994, pp. 213–218.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

APPENDIX: HYPERPARAMETER TUNING RESULTS

Loss	Batch size	Drop-out	Epochs	#neurons	Learning Rate	Ratio V-T	Time
7.92	4096	0.0103	1418	1906	0.00086	0.849	981
8.13	256	0.0116	906	228	0.00034	0.872	679
8.35	512	0.0130	681	201	0.00035	0.755	343
8.50	2048	0.0068	677	580	0.00030	0.947	262
8.55	2048	0.0071	1193	566	0.00038	1.098	461
8.65	4096	0.0055	1065	1511	0.00072	1.092	632
8.68	256	0.0011	1358	758	0.00013	2.761	1017
8.71	512	0.0016	1047	1564	0.00021	2.313	730
8.72	1024	0.0105	1244	520	0.00015	1.009	500
8.72	2048	0.0086	1272	779	0.00096	1.077	510
8.73	256	0.0042	1570	1852	0.00075	1.652	1725
8.74	1024	0.0102	763	479	0.00055	0.992	309
8.78	512	0.0103	803	794	0.00002	0.841	415
8.79	4096	0.0036	691	1969	0.00088	0.998	490
8.83	4096	0.0094	1291	1305	0.00028	1.085	698
8.84	2048	0.0079	1318	1184	0.00064	1.022	608
8.84	4096	0.0108	1272	630	0.00090	0.933	594
8.88	4096	0.0122	1352	1110	0.00040	1.000	718
8.91	2048	0.0092	1320	1044	0.00010	1.062	565
8.92	512	0.0019	1039	596	0.00027	2.049	501
8.94	512	0.0085	683	1435	0.00096	1.071	442
8.97	512	0.0182	950	1669	0.00098	0.889	694
8.98	1024	0.0144	788	880	0.00016	0.926	317
8.99	4096	0.0137	956	1221	0.00018	0.892	527
8.99	2048	0.0095	635	1403	0.00039	1.019	328
9.00	4096	0.0146	745	744	0.00046	0.780	353
9.02	1024	0.0108	610	1465	0.00095	0.886	346
9.05	1024	0.0134	1473	530	0.00037	1.074	586
9.05	4096	0.0032	854	686	0.00035	1.202	389
9.05	2048	0.0014	796	1266	0.00014	1.784	386
9.07	512	0.0154	770	1411	0.00094	0.947	473
9.07	2048	0.0173	888	1397	0.00097	0.832	448
9.08	2048	0.0044	963	1245	0.00066	1.464	442
9.09	4096	0.0035	604	1955	0.00097	0.972	440
9.10	1024	0.0016	3177	711	0.00011	2.595	1186
9.11	1024	0.0183	1220	1471	0.00079	0.953	691
9.11	2048	0.0050	1393	1455	0.00045	1.309	737
9.13	2048	0.0112	1061	913	0.00086	1.046	434
9.13	4096	0.0103	858	1120	0.00018	0.912	438
9.15	1024	0.0137	443	680	0.00009	0.742	177
9.15	4096	0.0123	688	1887	0.00083	0.862	484
9.15	4096	0.0154	1473	1157	0.00052	0.970	786
9.17	1024	0.0030	1600	1363	0.00098	1.862	870
9.18	2048	0.0041	821	931	0.00096	1.364	330
9.18	2048	0.0149	1477	1410	0.00015	1.090	729
9.18	4096	0.0024	773	1290	0.00023	1.445	419
9.19	1024	0.0163	778	404	0.00042	0.840	324
9.19	512	0.0104	805	975	0.00010	1.137	406
9.20	4096	0.0187	1232	1076	0.00011	0.761	603
9.21	1024	0.0159	309	376	0.00074	0.701	126

9.21	2048	0.0048	568	1873	0.00027	1.101	384
9.23	2048	0.0071	807	726	0.00042	1.143	320
9.23	2048	0.0101	1059	1917	0.00091	0.950	726
9.25	1024	0.0150	788	772	0.00031	0.956	297
9.26	2048	0.0074	879	1037	0.00097	1.180	378
9.27	512	0.0195	860	723	0.00010	0.855	443
9.28	1024	0.0131	516	820	0.00003	0.719	203
9.28	4096	0.0158	1402	557	0.00013	0.776	702
9.29	512	0.0154	1304	644	0.00093	1.109	639
9.30	2048	0.0050	1129	1441	0.00071	1.366	576
9.32	256	0.0092	1905	577	0.00020	1.490	1440
9.33	2048	0.0058	993	714	0.00019	1.137	358
9.34	2048	0.0031	959	1288	0.00045	1.563	447
9.35	256	0.0196	963	589	0.00038	0.960	715
9.36	2048	0.0098	716	1195	0.00050	1.068	316
9.37	512	0.0151	652	1756	0.00087	0.973	495
9.38	512	0.0116	905	816	0.00065	1.178	447
9.39	1024	0.0148	1128	1878	0.00090	0.967	783
9.41	2048	0.0023	1413	1650	0.00072	2.149	829
9.41	512	0.0136	229	695	0.00094	0.837	115
9.41	1024	0.0102	1412	610	0.00051	1.285	538
9.43	1024	0.0175	1441	1129	0.00094	1.026	680
9.43	512	0.0046	1225	1866	0.00046	1.829	980
9.44	2048	0.0193	513	1841	0.00023	0.797	334
9.44	4096	0.0042	1369	1646	0.00027	1.487	861
9.44	512	0.0066	1341	1767	0.00060	1.586	1056
9.44	1024	0.0198	1025	1113	0.00094	0.951	474
9.45	256	0.0281	495	652	0.00048	0.763	369
9.45	512	0.0180	619	1870	0.00012	0.991	506
9.45	512	0.0019	4715	525	0.00041	2.939	2458
9.46	2048	0.0014	1230	747	0.00067	2.091	464
9.47	2048	0.0100	792	1063	0.00030	1.042	338
9.48	256	0.0225	260	533	0.00066	0.754	194
9.48	512	0.0143	1114	1061	0.00052	1.185	588
9.49	2048	0.0105	1358	946	0.00094	1.224	509
9.49	1024	0.0163	1292	1825	0.00094	0.957	913
9.49	4096	0.0189	1355	1187	0.00036	0.892	749
9.50	2048	0.0194	792	1564	0.00053	0.674	430
9.51	512	0.0133	1324	1587	0.00095	1.198	942
9.51	1024	0.0110	1042	1044	0.00053	1.262	445
9.52	512	0.0145	1324	1426	0.00021	1.253	825
9.52	1024	0.0146	1150	667	0.00063	1.073	422
9.53	2048	0.0111	1448	931	0.00046	1.231	586
9.53	2048	0.0187	1065	1837	0.00067	0.904	708
9.53	4096	0.0089	633	1181	0.00038	0.951	327
9.53	2048	0.0051	532	530	0.00039	1.109	191
9.54	2048	0.0195	941	1357	0.00039	0.938	473
9.54	1024	0.0122	1070	885	0.00082	1.195	447
9.54	1024	0.0014	3511	795	0.00024	2.938	1423
9.54	4096	0.0075	1227	1781	0.00090	0.980	841
9.54	1024	0.0025	1966	1495	0.00063	2.386	1100

9.54	512	0.0241	295	515	0.00095	0.713	151
9.54	2048	0.0115	1155	1593	0.00070	1.117	660
9.55	4096	0.0182	586	946	0.00080	0.758	284
9.56	512	0.0053	1486	1014	0.00039	1.836	760
9.57	512	0.0021	1315	1058	0.00024	2.414	714
9.58	1024	0.0028	1137	1099	0.00058	2.039	523
9.58	1024	0.0037	1351	1345	0.00034	1.902	733
9.60	1024	0.0144	923	878	0.00086	1.079	378
9.61	512	0.0098	1354	526	0.00064	1.358	679
9.61	512	0.0274	668	992	0.00059	0.843	345
9.61	1024	0.0128	905	1632	0.00060	1.183	540
9.62	2048	0.0027	503	1779	0.00053	1.337	322
9.62	512	0.0116	1232	752	0.00026	1.291	633
9.63	256	0.0046	1132	1745	0.00016	1.939	1147
9.64	512	0.0336	408	768	0.00048	0.706	215
9.65	1024	0.0141	779	1132	0.00097	1.044	347
9.65	512	0.0185	675	1059	0.00041	1.008	339
9.65	512	0.0071	1171	912	0.00042	1.557	570
9.65	512	0.0227	674	194	0.00050	0.742	341
9.68	1024	0.0176	1035	1645	0.00068	1.037	666
9.69	4096	0.0078	961	1317	0.00061	1.197	551
9.69	1024	0.0103	1147	811	0.00032	1.287	454
9.69	256	0.0277	642	449	0.00025	0.793	476
9.69	1024	0.0116	1148	1601	0.00093	1.300	705
9.69	512	0.0136	585	1259	0.00043	1.116	328
9.71	4096	0.0120	1495	987	0.00064	1.107	719
9.72	256	0.0179	977	822	0.00010	1.068	738
9.72	4096	0.0023	863	795	0.00040	1.364	418
9.73	512	0.0026	3663	829	0.00014	2.670	1876
9.76	2048	0.0056	1358	1024	0.00066	1.513	560
9.77	512	0.0118	1393	1786	0.00053	1.374	1086
9.78	256	0.0013	4018	1386	0.00040	3.475	3436
9.79	1024	0.0184	881	1673	0.00020	1.060	561
9.79	4096	0.0117	1145	1912	0.00097	0.797	801
9.79	4096	0.0176	646	1405	0.00058	0.825	350
9.80	4096	0.0091	1292	1126	0.00060	1.154	647
9.80	4096	0.0120	631	1236	0.00092	0.920	325
9.81	512	0.0056	971	1278	0.00040	1.645	559
9.81	4096	0.0186	734	1985	0.00054	0.809	544
9.82	1024	0.0019	3403	672	0.00096	2.983	1390
9.83	512	0.0061	703	1132	0.00082	1.529	368
9.85	4096	0.0189	1478	1977	0.00049	0.951	1087
9.85	512	0.0130	828	1039	0.00060	1.196	426
9.85	1024	0.0147	1320	1253	0.00070	1.227	687
9.85	512	0.0149	519	894	0.00057	1.043	269
9.85	1024	0.0127	931	1211	0.00036	1.190	470
9.87	1024	0.0122	651	1545	0.00073	1.147	384
9.88	1024	0.0084	1342	1357	0.00022	1.567	724
9.88	512	0.0200	1300	1481	0.00076	1.102	858
9.89	1024	0.0125	831	977	0.00039	1.173	356
9.90	256	0.0177	103	945	0.00042	0.724	78

9.94	1024	0.0157	1420	1706	0.00020	1.268	918
9.95	1024	0.0185	875	1198	0.00075	1.024	453
9.95	1024	0.0047	1308	1208	0.00066	1.799	645
9.97	1024	0.0141	1351	1964	0.00012	1.315	1034
9.97	256	0.0361	656	638	0.00005	0.674	484
9.98	4096	0.0125	524	1680	0.00036	0.913	337
10.02	2048	0.0049	503	1808	0.00076	1.022	321
10.05	512	0.0309	607	225	0.00016	0.649	315
10.05	2048	0.0130	952	1242	0.00075	1.044	456
10.06	2048	0.0111	827	1697	0.00058	1.077	499
10.07	4096	0.0052	1105	1057	0.00063	1.295	572
10.08	1024	0.0164	1327	1600	0.00021	1.228	800
10.11	512	0.0116	1463	1972	0.00016	1.490	1317
10.12	4096	0.0077	923	1549	0.00065	1.135	572
10.13	256	0.0135	2635	1221	0.00044	1.477	2121
10.14	4096	0.0109	1227	1867	0.00058	1.175	845
10.14	4096	0.0185	822	1897	0.00081	0.786	574
10.14	2048	0.0127	923	1464	0.00073	1.030	501
10.15	1024	0.0162	1479	1675	0.00063	1.244	960
10.16	512	0.0015	4397	1666	0.00040	3.699	3244
10.18	512	0.0148	509	382	0.00003	0.680	251
10.22	256	0.0234	150	413	0.00014	0.648	113
10.22	1024	0.0165	3996	826	0.00018	1.489	1564
10.28	4096	0.0166	806	1132	0.00085	0.926	394
10.31	256	0.0402	222	794	0.00015	0.627	165
10.34	1024	0.0148	1329	1465	0.00034	1.316	750
10.36	256	0.0388	205	591	0.00088	0.653	156
10.46	1024	0.0182	240	422	0.00016	0.680	99
10.48	512	0.0174	1486	1345	0.00027	1.304	913
10.51	256	0.0029	4573	1540	0.00027	2.930	4303
10.60	1024	0.0190	1268	1809	0.00018	1.227	865
10.62	512	0.0305	978	766	0.00002	0.728	496
10.62	4096	0.0086	724	1294	0.00082	1.021	393
10.63	1024	0.0056	4784	1079	0.00043	2.569	2007
10.66	1024	0.0181	2242	1607	0.00024	1.465	1365
10.68	1024	0.0169	771	1243	0.00094	0.888	402
10.69	1024	0.0464	960	423	0.00021	0.664	380
10.72	4096	0.0199	543	1776	0.00082	0.832	351
10.75	256	0.0173	594	366	0.00001	0.679	458
10.78	512	0.0451	425	533	0.00087	0.690	216
10.79	2048	0.0095	650	1369	0.00081	1.079	331
10.88	4096	0.0185	586	752	0.00096	0.889	260
10.89	1024	0.0427	924	938	0.00058	0.795	404
10.90	512	0.0406	663	935	0.00037	0.780	344
10.91	1024	0.0116	607	527	0.00002	0.746	237
11.27	512	0.0185	3795	1472	0.00030	1.817	2587
11.30	512	0.0461	877	702	0.00007	0.703	457
11.46	1024	0.0290	285	875	0.00006	0.668	123
11.76	512	0.0550	810	627	0.00029	0.734	407
11.76	512	0.0496	455	443	0.00020	0.670	226
11.78	2048	0.0195	764	1965	0.00070	0.938	550

11.79	1024	0.0238	156	691	0.00054	0.741	61
12.68	1024	0.0252	639	193	0.00021	0.811	244
12.76	1024	0.0149	289	681	0.00003	0.762	111
13.01	1024	0.0460	666	992	0.00003	0.709	289
13.46	256	0.0758	477	662	0.00019	0.681	375
13.61	1024	0.0305	817	140	0.00057	0.903	323
14.29	256	0.0234	520	129	0.00005	0.797	388
14.46	256	0.0830	368	408	0.00016	0.650	284
14.84	256	0.0427	295	350	0.00003	0.668	222
14.86	512	0.0805	203	525	0.00044	0.641	103
15.04	1024	0.0180	181	779	0.00003	0.773	70
15.09	512	0.0803	819	763	0.00026	0.791	427
15.18	1024	0.0113	149	326	0.00009	0.809	61
15.22	256	0.0377	766	309	0.00001	0.702	568
15.27	512	0.0884	399	984	0.00018	0.700	212
15.39	1024	0.0687	925	645	0.00007	0.759	366
15.61	1024	0.0474	377	646	0.00003	0.693	146
16.12	1024	0.0407	673	161	0.00015	0.831	258
16.17	1024	0.0900	488	829	0.00024	0.724	203
17.00	256	0.1037	213	582	0.00073	0.701	160
17.16	256	0.0100	470	102	0.00002	0.816	352
17.49	1024	0.0184	241	193	0.00007	0.800	94
17.77	512	0.0959	868	401	0.00009	0.754	409
18.28	256	0.0959	846	368	0.00006	0.768	619
18.50	1024	0.1157	773	556	0.00086	0.783	317
18.74	1024	0.0683	646	183	0.00036	0.869	249
18.82	256	0.0340	585	156	0.00002	0.770	443
19.07	256	0.0544	156	726	0.00002	0.738	117
19.24	1024	0.0219	833	125	0.00003	0.772	323
19.28	256	0.1074	404	514	0.00049	0.819	302
20.93	1024	0.0958	284	529	0.00010	0.739	113
21.68	1024	0.1270	564	803	0.00044	0.848	225
21.77	256	0.1230	555	257	0.00043	0.866	424
22.42	1024	0.0933	701	205	0.00009	0.807	269
22.55	512	0.0722	934	313	0.00001	0.801	478
22.89	512	0.0562	421	119	0.00009	0.869	207
23.82	512	0.1151	504	410	0.00007	0.842	251
24.10	256	0.1023	852	166	0.00005	0.875	645
24.22	256	0.1415	203	560	0.00055	0.843	152
25.58	1024	0.1513	748	312	0.00041	0.894	294
25.61	256	0.0635	532	213	0.00001	0.802	391
26.32	256	0.1588	565	926	0.00008	0.907	433
26.78	1024	0.1037	604	202	0.00008	0.874	244
27.34	1024	0.1159	805	797	0.00002	0.906	332
27.36	256	0.0853	532	272	0.00001	0.836	390
28.08	1024	0.1268	922	199	0.00011	0.941	362
28.22	512	0.0518	100	393	0.00001	0.791	51
28.39	1024	0.0995	864	322	0.00002	0.867	351
29.32	1024	0.1013	561	507	0.00002	0.886	220
29.84	512	0.1602	219	827	0.00009	0.875	115
31.17	512	0.1376	187	413	0.00005	0.826	90

32.02	1024	0.1021	235	100	0.00017	0.889	91
33.13	256	0.1951	225	929	0.00007	0.928	170
33.86	1024	0.0792	200	217	0.00001	0.814	79
34.07	1024	0.1454	880	648	0.00002	1.037	358
35.30	512	0.1691	222	367	0.00007	0.892	110
35.33	512	0.1899	866	311	0.00062	1.143	429
36.68	1024	0.1930	415	791	0.00007	0.994	168
39.03	256	0.0016	3444	1917	0.00017	0.546	1
40.97	1024	0.1668	108	463	0.00002	0.889	43
87.95	2048	0.0031	1422	1241	0.00044	0.757	1